Multi-Target User Interface design and generation using Model-Driven Engineering

THESIS

to obtain the degree of
MASTER OF SCIENCE IN COMPUTER SCIENCE
and MASTER OF SCIENCE IN HUMAN MEDIA INTERACTION
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, the Netherlands

by

M. (Mark) Oude Veldhuis
mark@oudeveldhuis.nl

May 3, 2013
Enschede, the Netherlands

Supervisors from the University of Twente
dr. E.M.A.G. (Betsy) van Dijk
dr. L. (Luís) Ferreira Pires
Dr.-Ing. C.M. (Christoph) Bockisch

Supervisor from Sigmax
ir. R.L.M. (Robert) Spee

Preface

This document is my thesis for the master studies *Computer Science* and *Human Media Interaction* from the University of Twente, the Netherlands. It describes the approach and results of my master assignment, which I conducted at Sigmax in Enschede. The master assignment consists of two related parts, one for each master programme.

I would like to use this opportunity to thank Luís, Betsy and Ivan, my supervisors from the university, for guiding me throughout the entire process and providing me with feedback, and Christoph for jumping in as third supervisor. Thanks to my colleagues at Sigmax; Rick, Danny and Robert in particular for having me as their roommate. Thank you Robert for being my supervisor at Sigmax and for your guidance. I appreciated your input a lot and learned many things, not only related to the master assignment.

Many thanks to my parents Joep and Gerda, and brother Sander. I appreciate your never-ending support during all my years of study. And let me reassure you, they have all been worth their while ;-)

A word of thanks to all members of study tour committee *Noodle*, of which I was the chairman. Marijn, David, Yme, Lex and Nils, my fellow committee members, my friends, thank you for being patient and coping with the fact that I was only one day per week at the university during most of our preparations, and many, many thanks for your commitment during the organization. Our amazing journey through South Korea, China and Hong Kong was worth every struggle and time consuming moment.

I would like to thank my fellow students Tim Harleman and Martijn Adolfsen. We studied together before and at the university, attended most of the courses together, and had lots of fun doing so.

Finally, I would like to thank all my friends whom I did not mention by name here and last, but most certainly not least, my sweet girlfriend Lysette for her support and love.

— Mark Oude Veldhuis
May 3, 2013

Abstract

The rapid development and wide spread adoption of smartphones and tablets creates a desire to deploy applications to multiple platforms without developing the same application for every target platform. Deploying an application to multiple platforms is challenging because of different operating systems, screen sizes, and device capabilities such as the presence or absence of a hardware keyboard. Additionally, the design of user interfaces is often based on experience and intuition, instead of explicit guidelines.

In this thesis we explore how a Model-Driven Engineering (MDE) environment can be developed that generates mobile applications for multiple target platforms, based on a single source model, while taking into account a set of user interface design guidelines. The design guidelines were developed based on an extensive literature study, expert interviews, field studies and a lab study. Seven guidelines were developed, of which six focus on design principles and high-level application behavior, and one focuses on navigation through hierarchical lists.

We developed a proof-of-concept MDE environment that takes a single source model as input and transforms it to an Android application for both smartphones and tablets. The guidelines that were incorporated were *be consistent*, *provide understandable feedback*, and *be supportive and minimize manual input*. Expert interviews with software architects and developers confirmed that such an approach can be helpful in the development of mobile applications in order to decrease development time and manage complexity. Adopting an MDE environment in an existing development environment was also seen as a challenging task. The flexibility of MDE is a great advantage, but also creates challenges. Depending on the context of use it can be difficult to determine the amount and abstraction level of metamodels to create.

Abbreviations

ATL	**A**TLAS **T**ransformation **L**anguage
EMF	**E**clipse **M**odeling **F**ramework
EMP	**E**clipse **M**odeling **P**roject
MBUID	**M**odel-**B**ased **U**ser **I**nterface **D**evelopment
MDA	**M**odel-**D**riven **A**rchitecture
MDE	**M**odel-**D**riven **E**ngineering
OMG	**O**bject **M**anagement **G**roup
PDA	**P**ersonal **D**igital **A**ssistant
PIM	**P**latform-**I**ndependent **M**odel
PSM	**P**latform-**S**pecific **M**odel
UI	**U**ser **I**nterface
UML	**U**nified **M**odeling **L**anguage

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an introduction to our research by giving our motivation, our problem statement, our goals, our approach and the document structure.

1.1 Motivation

We can no longer think of a society without mobile devices, smartphones and tablets in particular. It seems that everyone has at least a smartphone nowadays. These devices are used for communication with other people, but also to retrieve information from a central location using a mobile internet connection. Think of Wikipedia, or news sources such as CNN.

Mobile devices are also widely used in professional environments. When a police officer stops someone that violated the law, filing out all information about the offense and violator, as well as printing the ticket is often done using a mobile device. The data are sent to a remote server to make it available for further (automatic) processing. In daily life, one may find police officers, train conductors, parking attendants, waiters on terraces, and many other professionals that use mobile devices to make their jobs easier.

The variety in mobile devices in terms of dimensions, operating system, sensor capabilities, and so on, makes it difficult to develop a single set of user interface (UI) design guidelines that ensure a consistent user experience for the same application deployed on different devices. The same issue is relevant for the technical development of mobile applications, in terms of keeping the development time of new applications short, without having to develop a completely new application for each different device.

This thesis aims to explore the possibilities to address these two issues, i.e. developing a set of UI guidelines that are applicable to multiple target platforms and developing a single development environment that can be used to build an application for multiple targets, from a single model.

1.2 Problem statement

Our problem statement was inspired by practical problems at Sigmax, but can be generalized. Sigmax is a company located in Enschede, the Netherlands, and develops mobile solutions for law enforcement, supervision, field service and inspection. They deliver complete solutions using mobile devices — such as smartphones and tablets — and back-end solutions. The problem addressed by this research is limited to the mobile devices.

The design and development of user interface as well as other software components can be improved in terms of consistency and development time. Comparable user interfaces and software components that could be re-used, are often re-developed. This approach results in time-consuming development, inconsistent user interfaces and a high rate of errors. The rapid development and wide spread adoption of mobile devices nowadays creates a desire to deploy the same application to multiple target platforms.

Additionally, the UI design of mobile applications is often based on intuition and experience, instead of explicit guidelines. Although feedback is sometimes obtained from users, a structured way of questioning end-users and formal validation to assess whether the user interfaces offer a pleasant user experience is often missing.

1.3 Goals

The first goal of this work is to deliver a proof of concept that shows that a Model-Driven Engineering environment can be developed, to automatically generate applications for different mobile devices types, such as smartphones and tablets. By modeling mobile applications and re-using these models, the consistency between mobile applications and their user interfaces potentially increases, and the development time is reduced.

The second goal of this work is to develop a set of user interface design guidelines that should be taken into account during the design and development of mobile applications. The Model-Driven Engineering environment that was developed as a proof of concept was supposed to incorporate these guidelines.

To achieve these goals, the following main research question was stated:

> **RQ.** *How can a Model-Driven Engineering environment be developed to increase the consistency, usability and development speed of mobile applications, while taking into account user interface design guidelines?*

In order to answer the main research question, the following sub questions were stated:

> **SQ1.** *How can Model-Driven Engineering be applied during the development of mobile applications, in order to speed up development and cope with a variety of mobile devices?*

> **SQ2.** *Which are the challenges in user interface design and what are the usability issues for mobile applications?*

SQ3. *Which user interface design guidelines should be taken into account during the development of mobile applications for different target platforms?*

1.4 Approach

To answer the research questions, the following steps were taken. The research questions answered in each step is denoted between parentheses:

1. An extensive literature study was conducted to gain background knowledge on Model-Driven Engineering (MDE) and in particular MDE applied to user interfaces and mobile applications. (SQ1)

2. An extensive literature study was conducted to gain background knowledge on User Interface for mobile application design. The literature study identified several common challenges and common design guidelines for mobile user interface design. (SQ2)

3. At Sigmax we conducted expert interviews and field studies to gain a better understanding of the current challenges and issues of Sigmax applications. A lab study was conducted to investigate how people prefer to browse hierarchical data structures on smartphones and tablets. (SQ2)

4. Several design guidelines were developed. The joint issues and challenges found during the literature study, expert interviews, field studies and lab study were the primary source for the development of these design guidelines. (SQ3).

5. A proof-of-concept was developed to answer the main research question. The knowledge gained during the literature study on MDE was combined with the design guidelines in order to develop a proof-of-concept that can generate mobile applications for different mobile devices, based on a single source model. (SQ1, RQ)

6. The developed proof-of-concept was demonstrated to experts in software architecture and development at Sigmax. Afterwards, semi-structured interviews were conducted to assess whether such an approach helps to increase the consistency, usability and development speed. (RQ)

1.5 Document outline

The remainder of this thesis is organized as follows:

Chapter 2 provides background knowledge on Model-Driven Engineering. The purpose of this chapter is to provide the required understanding of MDE principles, techniques and viewpoints.

Chapter 3 provides background knowledge on user interface design for mobile devices. The purpose of this chapter is to provide an overview of common challenges in user interface design for mobile devices, and to determine which data gathering techniques are useful to identify current issues with mobile devices at companies like Sigmax.

Chapter 4 presents the data gathering techniques that we applied to identify the current issues with mobile devices at Sigmax and the results that we found. The purpose of this chapter is to show how the results from the data gathering techniques were used as input for user interface design guidelines, and which design guidelines were developed.

Chapter 5 provides the solution design for the proof of concept. The purpose of this chapter is to show the solution design of a Model-Driven Engineering environment that allows the generation of applications for different mobile devices. This chapter also includes an overview of the used tools and languages.

Chapter 6 assesses the acceptance of this research by implementing a case study using the proof of concept. The case study elaborates on how the solution design is technically implemented, and shows that the proof of concept works. This chapter also provides the results from expert interviews that were used to assess the practical acceptance of this research.

Chapter 7 provides the final conclusion of this work, answers the research questions and identifies topics for future work.

Chapter 2

Model-Driven Engineering

This chapter provides background knowledge on Model-Driven Engineering. The purpose of this chapter is to provide the required understanding of MDE principles, techniques and viewpoints.

2.1 History and introduction

The first Fortran compiler was delivered in 1957 and allowed programmers for the first time to specify *what* the machine should do rather than *how* it should do that. The recent research, activities and other developments related to Model-Driven Engineering are a natural continuation of the trend where software development moves from a code-centric to model-based practice [4, 6].

In 2000, the Object Management Group started the Model-Driven Architecture (MDA) initiative [6] and in 2001 the MDA guide was adopted [26]. MDA describes a software development approach where abstract models of software systems are created, and then transformed to concrete implementations [18]. MDE is a software development approach that combines *process* and *analysis* with the MDA principles and techniques [18, 32]. MDE may be seen as a generalization of MDA [35].

2.2 Models, models and models

Models are the primary artifacts of model-driven development and are abstractions of some aspect of a system [18]. They are described in a well-defined modeling language, described in a *metamodel*, as for example the Unified Modeling Language (UML). Typically, model descriptions are graph-based and rendered visually [34]. UML is however tightly bound to programming languages — as UML was originally meant for representing code — making it less useful to provide domain-specific modeling capabilities [25].

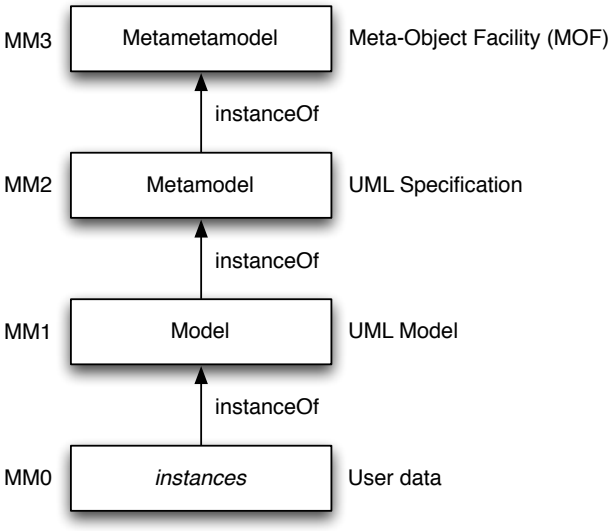

FIGURE 2.1: Metamodeling levels in Model-Driven Development [4]

Metamodels. A metamodel defines the language a model is written in, making it a model of a model [18, 34]. Since a metamodel is also a model itself, a *metametamodel* defines the language a metamodel is written in. We could continue this recursion infinitely, but that would not be practical. Figure 2.1 shows that MDE typically recognizes four levels of metamodeling. The Meta-Object Facility (MOF) is a metametamodeling language that is closely related to the UML and is used to define the abstract syntax of modeling languages [4, 18, 44]. An alternative for the MOF is, for example, *Ecore*, from the Eclipse Modeling Project [15].

Viewpoints. An integral part of Model-Driven Architecture is its viewpoints: the *computation independent model viewpoint*, the *platform-independent viewpoint* and the *platform-specific viewpoint*. The computation independent model is used to model the requirements for a system. In this work we however focus on the other viewpoints, as modeling requirements is not within the scope of this research.

The platform-independent viewpoint focuses on the operation of a system, but hides technical details necessary for a particular platform [26]. This viewpoint focuses on aspects that are unlikely to change for multiple platforms [18]. A platform-independent model (PIM) is a specification of the system under development, derived from a platform-independent viewpoint [44]. Such a model is constructed using a well-defined platform-independent modeling language in order to fulfill system application requirements [26].

A *platform-specific model* (PSM) is often derived from a PIM and adds technical details that specify the system for a particular platform [18, 26]. It is a representation of a system from a *platform-specific viewpoint* [26]. A PSM specifies the system under development for a particular platform, expressed in a well-defined modeling language [26, 44]. Since programming languages are structurally well-defined modeling languages, source code can also be seen as a special case of PSM.

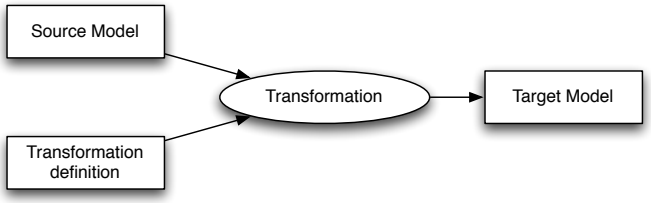

FIGURE 2.2: Generic representation of model transformations in MDA [26]

2.3 Transformations

Transformations are an integral and critical part of Model-Driven Engineering [26, 31, 34]. Functionality specified in a PIM may be transformed to a PSM via some transformation definition, or mapping [26, 32]. The complexity of bridging the conceptual gap between problem and implementation domain is dealt with using possibly automated transformations [18].

The typical transformation in MDE is from a PIM to PSM [32]. The PIM and other relevant information are combined in the transformation process that produces a PSM [26]. An abstract model of a transformation is shown in Figure 2.2. The *source model* and *transformation definition* are both input for the transformation process. The transformation definition describes how the *target model* can be obtained from the source model.

There exist two kinds of transformations in MDE: *model-to-model* and *model-to-text transformations*. We discuss them in more detail below.

Model-to-model transformations. A popular transformation language is the ATLAS Transformation Language (ATL) [31]. ATL offers a declarative syntax to define rules that contain patterns to match instances from a source model, as well as a target pattern to create the target models for each of the matched instances. Imperative transformation code may be defined in ATL and can contain a declarative target pattern as well as an action block, which is nothing more than a sequence of statements. Listing 2.1 shows an example of a simple ATL transformation rule.

```
1 rule Class2Table {
2  from c: Class!Class
3  to t: Relational!Table (
4   name <- c.name
5  )
6 }
```

Listing 2.1: A simple ATL rule that transforms classes to relational tables. The name of the *Table* is set to the name of the *Class*.

Figure 2.3 shows the operational context of ATL.

Model-to-text transformations. Another kind of transformation are model-to-text transformations, which are able to generate application source code, for example. An example of a model-to-text transformation language is the Xpand language, developed in the scope of the *Eclipse Modeling Framework* (EMF). It was originally part of *openArchitectureWare* before it became a component of the EMF. Xpand uses templates that contain transformation definitions.

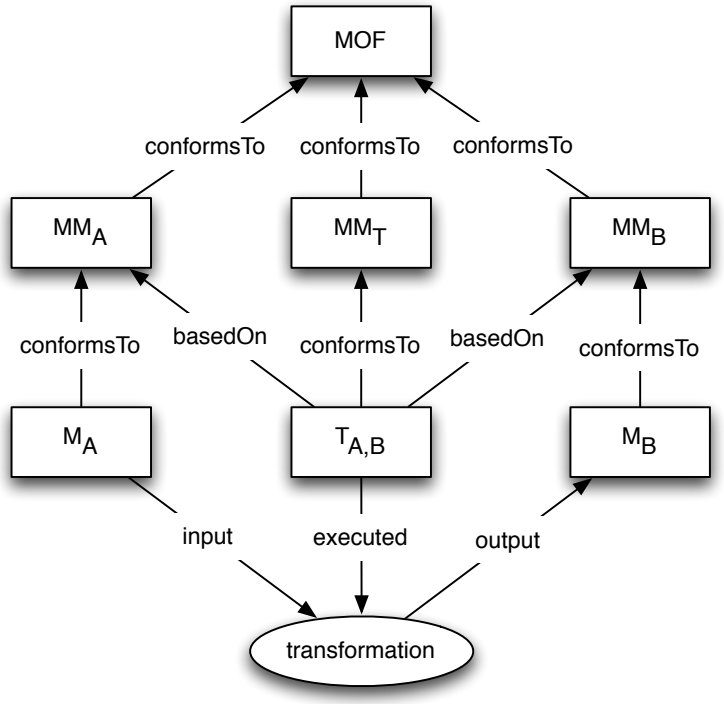

FIGURE 2.3: The operational context of ATL [29, 31]. M_A, M_B and $T_{A,B}$ are respectively the source, target and transformation models. These models conform to the metamodels MM_A, MM_B and MM_T, respectively. For ATL, MM_T is the ATL metamodel.

These definitions can invoke other definitions, and allow the use of for-loops and if-statements, amongst others [45]. Listing 2.2 shows a simple transformation template with two definitions. Model-to-text transformations create files that are filled with template code, in which the details are derived from models.

```
1  <<DEFINE Root FOR data::DataModel>>
2    <<EXPAND Entity FOREACH entity>>
3  <<ENDDEFINE>>
4  <<DEFINE Entity FOR data::Entity>>
5    <<FILE name + ".java">>
6      public class <<name>> {
7        <<FOREACH attribute AS a>>
8           private <<a.type>> <<a.name>>;
9        <<ENDFOREACH>>
10     }
11   <<ENDFILE>>
12 <<ENDDEFINE>>
```

Listing 2.2: A simple Xpand transformation template containing two definitions. Here, a *DataModel* is transformed into corresponding Java files.

2.4 Model-Based User Interface Development

Model-Based User Interface Development (MBUID) refers to the use of MDE viewpoints and principles during the development of user interfaces [37]. Parallel to the development of the MDE

approach, researchers have been investigating how MDE can be adopted for the development of user interfaces, to profit from the benefits offered by MDE.

Model-Based User Interface Development is a form of advanced user interface engineering [47]. With this approach, an abstract description of the UI is provided which is called a User Interface Model (UIM). According to Silva et al. [47], a UIM is composed of four models:

1. The **Application Model** describes application properties relevant to the UI. Examples include problem or domain models that describe concepts of the application. For a library application, a *book* is an example of a concept.

2. The **Task-Dialog Model** describes tasks that the application end-users should be able to perform, as well as the relationships between different tasks. Examples of tasks are browsing through a list of items, selecting one and performing a certain action on this item, such as editing or deleting it.

3. The **Abstract Presentation Model** provides a conceptual description of the structure and behavior of the user interface using abstract presentation objects such as *text labels* or *text input boxes*.

4. The **Concrete Presentation Model** describes the visual parts of a user interface in detail, and how the interface is composed of concrete presentation objects, or widgets. It represents the final structure and look and feel of the user interface, but is only available in an editor and must be transformed to a certain UI implementation language.

The relationship between different tools and models is visualized in Figure 2.4. First, the application and task-dialog models should be developed, both of which serve as input to an *abstract design tool*, providing an abstract presentation model as output. This model is used as input to a *concrete design tool*, which is able to produce a concrete presentation model based on guidelines.

Realizing this model-based approach calls for the support of tools. *Modeling tools* are (graphical) environments that support the creation of UI models. *Modeling assistants* are (graphical) environments that support UI developers by providing feedback to the developer. They provide features such as model checking and validation [47].

2.5 Cameleon Reference Framework

An implementation of a Model-Based User Interface Development approach is the Cameleon Reference Framework (CRF). It was developed to support the model-based development of user interfaces [19, 37, 55], and describes the development steps and adopts certain conceptual elements from the original MDA approach, such as different abstraction levels and model transformations. Figure 2.5 shows a simplified version of the CRF. This simplified version is sufficient to identify the core characteristics of the framework.

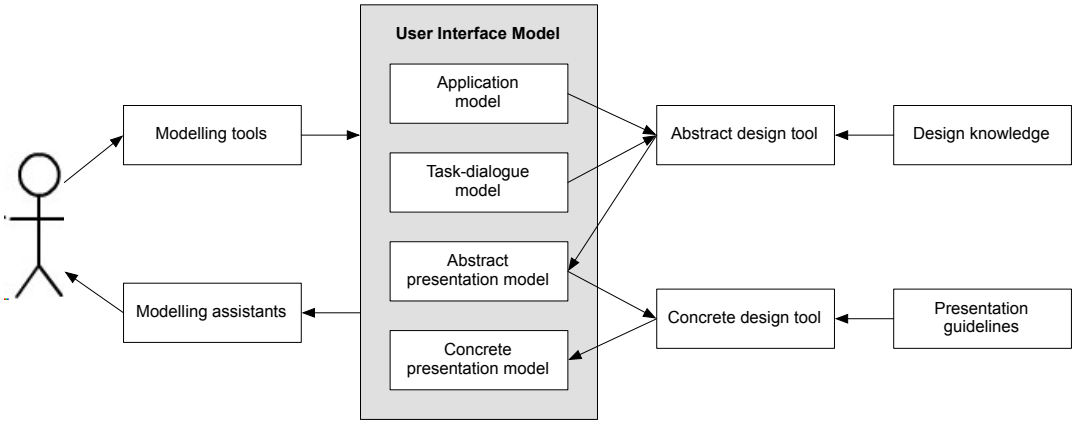

FIGURE 2.4: User Interface Engineering in MBUID [47]

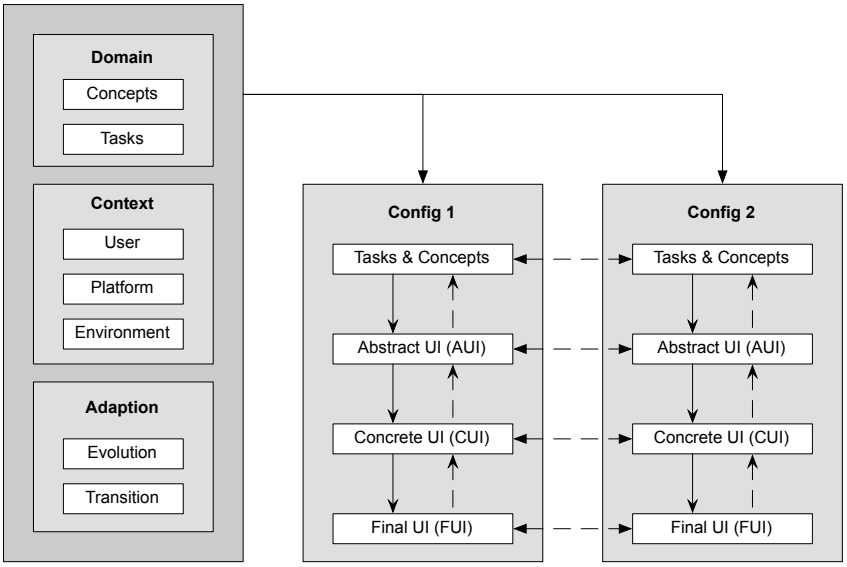

FIGURE 2.5: The (simplified) Cameleon Reference Framework [19, 37, 55, 57]

CRF can be seen as an implementation of the generic Model-Based User Interface Development approach we discussed earlier in Section 2.4.

2.5.1 Characteristics

In CRF, three metamodels known as *ontological models* identify the key dimensions necessary to solve a given problem [19]: *domain*, *context* and *adaptation* models, as shown in Figure 2.5. We discuss each characteristic in more detail, and how they comply with the generic Model-Based User Interface Development approach.

Domain Models provide a description of classes and objects that may be manipulated by users that interact with a system, within the domain of the system [19, 55]. They can be described using an UML class diagram, entity-relationship-attribute

model or some other object-oriented model [55]. Concepts are, for example, *objects* that need to be inspected in some *location*. Tasks could include *inspecting* or *editing* a certain object or location.

The *concepts* model can be mapped to the *application model* of the generic user interface engineering method as discussed in Section 2.4, since they share a common goal. The *tasks model* can be mapped to the *task-dialog model* of the generic approach.

Context Models specify the context of use based on three characterizing aspects [19, 55]: 1. *User* represents a stereotypical user of the system in terms of perceptual, cognitive and action capacities; 2. *Platform* is modeled in terms of resources such as available input and output mechanisms, screen size, memory size, operating system and other technical aspects; 3. *Environment* is the set of all aspects that may have an impact on the system or user aspects, such as the illumination of a certain environment. The boundaries of this set are usually determined by domain analysts so that only relevant aspects are included.

Adaptation Models are specified in terms of *evolution* and *transition*, so that the system is able to properly respond to a change of context of use [19]. The *evolution* model specifies to what implementation or configuration the system should switch in case of a change in context. For example, an evolution model can define what should happen when changes in brightness or screen orientation are detected. The transition model aims to soften changes between different contexts. It allows specification of a *prologue* that prepares the context transition, and an *epilogue* that restores system execution in the new context. During some transitions it is possible that the current activity is paused before the transition, and resumed as soon as the transition is finished.

2.5.2 Abstraction Levels

Figure 2.5 shows that CRF recognizes four levels of abstraction, which correspond with development steps. We discuss each of these four levels of abstractions below.

Tasks & Concepts (T&C). The tasks and concepts for a user interface immediately follow from the domain model in CRF, and describe domain concepts and tasks the user should be able to perform.

Abstract User Interface (AUI). An AUI describes how the interface is composed using *Abstract Interaction Objects (AIOs)* [46, 55], and groups tasks according to criteria. AIOs are available in two flavors: *components* and *containers*, where the latter either contains components or other containers. An AUI mainly describes the interface in terms of input and output components, but it does not include behavior.

The AUI model is related to the abstract presentation model from the generic user interface engineering approach depicted in Figure 2.4.

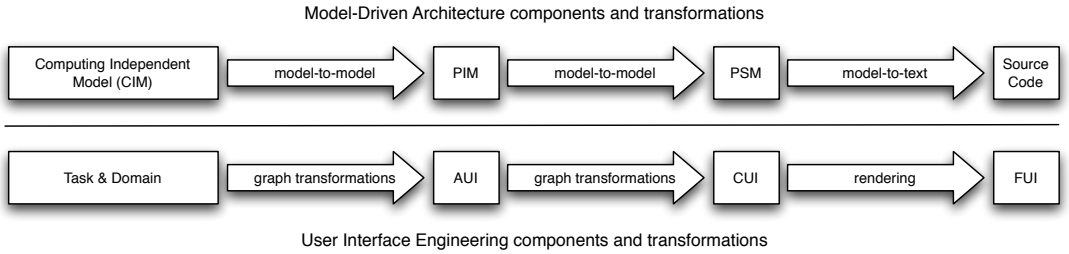

FIGURE 2.6: CRF correspondence to MDA [55]

Concrete User Interface (CUI). A CUI concretizes an AUI for a given context of use by means of *Concrete Interaction Objects (CIOs)* [46, 55]. Although a CUI makes the look and feel of a user interface explicit, it only runs in an MBUID environment. The context of use is based on the three factors described in Section 2.5.1.

The CUI model can be mapped to the concrete presentation model of the generic MBUID approach presented in Figure 2.4.

Final User Interface (FUI). An operational and running implementation of a CUI for a particular platform, i.e. source code.

2.5.3 Model-Driven Architecture correspondence

So far we only discussed the models in the Cameleon Reference Framework. To comply with MDA, transformations must be defined. Figure 2.5 depicts transformations with the arrows in the *config*-blocks. The downward, upward and horizontal arrows illustrate reification, abstraction and translation transformations, respectively.

The models used in CRF and MDA in general show similarities in both their purpose and interdependence [46, 55]. Both approaches start at a high abstract level by specifying what a user interface or system should be able to do. Then, an abstract model is created that specifies this functionality without taking into account the computing platform. The last two models are a concrete implementation where the platform is taken into account and the actual source code, which is the FUI for a MBUID.

Figure 2.6 shows how the models included with CRF approach correspond with the MDA approach defined by the OMG [55].

2.6 Mobile Devices

MOdel-Driven Engineering can be applied in the development of applications for mobile devices, and it is most likely also beneficial for larger applications. A metamodel could be adopted or defined that describes the target platform in order to generate a platform-specific model for any kind of device. However, there are specific challenges that arise when applying model-driven engineering for mobile devices, which we discuss in this section.

2.6.1 Challenges

When developing applications for mobile devices, the main problem is that different mobile devices often run different operating systems. Additionally, while mobile devices offer several possibilities they also have their limitations. We discuss the challenges in further detail.

Multi-Target Systems. A multi-target system is an interactive application that is aimed for use in multiple environments, has different input and output modalities, and is built for multiple computing platforms [53].

Concerning input and output modalities, different devices may differ in their capabilities. For example: some devices have GPS-sensor or camera in order to support user input, while other devices do not. The different screen sizes and aspect ratios are examples of differences in output modalities.

Different devices also vary in terms of available energy. While some devices can work without recharge for days under heavy usage, others need to be recharged after hours under the same heavy usage.

User Interfaces. In [55], Vanderdonckt recognizes that mobile computing platforms pose a challenge in UI design. One challenge follows from the differences in screen sizes and aspect ratios. An application that runs on a tablet may present the information differently than the same application on a smartphone, by exploiting the larger screen. Other challenges are, for example, the (un)availability of a hardware keyboard or other input and output modalities, such as GPS-sensors or speakers.

2.6.2 Approaches

In [33], Kim et al. applied Model-Driven Development during the development of an Android smartphone application. They specified a platform-independent model of the application, and used ATL to transform the PIM to a PSM. Both the PIM and PSM were UML models. The case study is a classic example of applying a model-driven approach, in which they only generated the code skeleton for the Android application. Application-specific code and logic still had to be implemented. They did however show that a model-driven approach is beneficial for the development of smartphone applications, as they conclude that by modifying the transformations, also iPhone and Windows Mobile versions of the generated code skeleton could be generated.

While their approach clearly illustrated the power of Model-Driven Engineering in the development of mobile applications, it remains a challenge to develop approaches that are more flexible, scalable and able to deal with the many differences between devices. The model of a system for different target devices remains the same, but the primary challenge is to provide proper transformations between the models, as illustrated in Figure 2.7. Querying Google Scholar for MDE applied for mobile devices yielded two generic approaches: colored graph transformations [53] and transformation templates [3]. Figure 2.7 shows that both approaches operate in the same operational context, namely in the transformational context from abstract to concrete model.

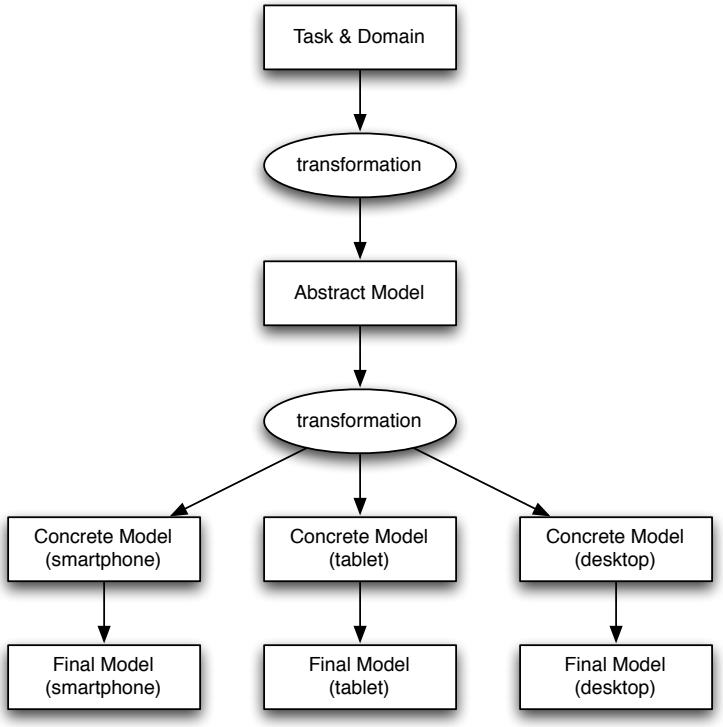

FIGURE 2.7: Transformational approach using transformation rules [3, 53]

In Section 2.3 we introduced ATL, which is a language to specify model-to-model transformations. One of the advanced features of this language is its support for rule inheritance. Rule inheritance allows the re-use of transformation rules, and is also a mechanism to support polymorphic rules [30]. Rule inheritance can be used to describe transformation rules that are common for different targets. These rules can then be extended in order to capture target specific transformations.

Listing 2.3 gives an abstract example of rule inheritance in ATL. Listing 2.4 shows the compiler interpretation of the example. In ATL, only the *to*-part of a sub-rule is unified with the *to*-part of its parent. Following this example, one could also define a rule *C* that extends rule B. The *to*-part of rule C would be the unification of the *to*-parts of rule A, B and C.

```
1  abstract rule A {
2   from [fromA]
3   using [usingA]
4   to [toA]
5   do [doA]
6  }
7  rule B extends A {
8   from [fromB]
9   using [usingB]
10  to [toB]
11  do [doB]
12 }
```

Listing 2.3: A simple abstract example of rule inheritance in ATL

```
1  rule B {
2   from [fromB]
```

```
3   using [usingB]
4   to [toA.bindings union toB.bindings]
5   do [doB]
6  }
```

Listing 2.4: Compiler interpretation of Listing 2.3

Chapter 3

User Interface design for mobile devices

This chapter provides background knowledge on user interface design for mobile devices. In this chapter we present an overview of common challenges in user interface design for mobile devices, and determine which data gathering techniques are useful to identify current issues in mobile devices at a company such as Sigmax.

3.1 Introduction

One of the goals of Human-Computer Interaction (HCI) design is to minimize the barrier between what a user wants to achieve, and the computer's ability to understand the task [49]. In order to minimize that barrier, we must know what causes it. Here, we investigate the problem areas and issues that arise during interaction with a mobile device application. This allows us to better understand the user needs.

3.2 Hardware challenges

Mobile devices are less powerful in terms of processing power and graphics than traditional desktop computers, often differ in form factors and are limited in the available peripherals [7]. Although the processing power of mobile devices is rapidly increasing, the form factors and peripherals remain challenges. The main issues that are related to Human-Computer Interaction and mobile devices, are input and output challenges.

Input. Hardware keyboards themselves are small, and have small keys because of the limited amount of space on a mobile device [11], which results in lower data-entry

rates [49]. Researchers first claimed that the data-entry rate would be comparable to that of desktop applications and error-rates would not increase. More recent research however showed that using a hardware keyboard can be quite tricky and cumbersome, especially for people with thick fingers [28].

Software keyboards, i.e. on-screen keyboards, can be used but often require the use of a stylus, which is a frequently used mechanism and acceptable alternative [28]. However, using a stylus has its disadvantages. People can lose it, have issues use it because of its small form factor, or using a stylus obstructs interaction with other graphical elements on the screen.

Different types of input also play a role. One may distinguish text and numerical data, but also dates or GPS coordinates, for example [42]. This introduces issues where a default text entry mechanism may not deliver the expected user experience while entering a date, time or geographical coordinate.

Output. One of the most obvious hardware issues of mobile devices related to output of information is the physical size of the screen [11, 42, 49]. Although there are devices with larger screens, they are not always desirable because of the required mobility [11, 28]. Switching between landscape and portrait mode introduces new possibilities in application design, but also issues such as determining in which situations to switch between these two modes [42].

Because of the limited screen size, a trade-off between what should and should not be displayed on the screen must be made, which can be achieved by conducting user experiments [28]. Other identifiable issues related to screens are the low resolution and lower amount of colors compared to regular computer screens, and the width to height ratio that often differs from the usual 4:3, 16:9 or 16:10 [7].

Lastly, although audio output is limited, it is still a very useful and suitable mechanism to notify users with a sound effect in case of an event [28].

3.3 Software challenges

The hardware limitations described in section 3.2 result in several software challenges.

Screen size. Limited screen sizes force developers to make a trade-off between what should and should not be on the screen, and result in the information being split up into several small presentation units [28]. This is especially challenging in case large amounts of data are available [1].

Hierarchical navigation. Navigation through hierarchical menus or other forms of structured information is difficult [1, 28]. Structuring this information is a challenge. With hierarchical menus one has to decide on the amount of levels and amount of items per level. Research shows that more hierarchical levels is better than more items per level, and that each level ideally should have 4 to 8 items [21, 28]. However, this particular research was conducted with outdated mobile devices.

Data input. The fact that text entry on mobile devices is slower than when using a regular computer keyboard [49], requires easy ways of entering information for different types of data such as text, numerical data, dates and locations. Using multimodal input here allows for easier input [42], such as, for example, a date picker to select a date, instead of manual entry.

3.4 Environmental challenges

Using mobile devices in professional environments may often result in the device being used outside, meaning that the weather outside becomes an important factor. As the device has to be used in cold weather as well, issues arise during interaction with a device while wearing gloves, for example, or reading from the screen on a bright sunny day.

Patchy sensors, poor or no network coverage or GPS-satellite reception can cause issues for mobile devices, especially in case the device uses contextual information to increase user experience [11].

Related to network coverage, the mobile Internet connection is often slow compared to more traditional wired or wireless connections, which can greatly affect interactivity when large amounts of data have to be retrieved from a remote source [7]. Business mobile Internet connections are also more expensive than mobile consumer Internet connections.

3.5 Design Guidelines from literature

The previous sections identified several challenges. These need to be taken into account when designing for mobile devices. The next step is to find solutions that address these issues. Several researches identified design guidelines that help HCI designers to develop user interfaces. These guidelines can be categorized based on their purpose. We identified *high-level* and *low-level guidelines*, both of which are explained further in this section.

3.5.1 High-level guidelines

From literature we identified several guidelines that we classified as *high-level guidelines*, which presents what should or should not be done in terms of interface design and application behavior. High-level guidelines do not specify exact design details.

Gong presents guidelines based on Shneiderman's *Golden Rules of Interface Design*, tailored to meet mobile interface design [22, 51]. Although published a while ago, in 2004, the guidelines are quite abstract and still applicable today. The work specifies a number of abstract guidelines, such as *offer informative feedback*, *reduce short-term memory load* and *design for enjoyment*. The guidelines describe points of attention that should be taken into account by a designer, and help to design an application from the perspective of the end-user.

In addition and related to these guidelines, Kukkonen and Kurkela developed seven principles for highly goal-driven mobile services [43]. They should allow for mobility, be useful, show relevant information, be easy to use, allow the most important information to be easily located, use user's terminology and their way of thinking, and finally they should adapt to every user's own needs.

In combination with the work of Love [36], we selected three high-level guidelines that are of importance: *consistency*, *context-awareness* and *multi-modal input*. We discuss these in more detail:

Consistency. In [36], Love states that it is important to offer the same user experience among different applications on mobile devices or the same application on different platforms, something that was also derived from Shneiderman his *Golden Rules of Interface Design* [22, 51].

Apple (iOS) [2], Google (Android) [23, 24] and Microsoft (Windows Phone) [38, 39] are major players on the mobile smartphone market. Their operating systems form the majority of the smartphone market [20]: Android runs on 69.7% of all smartphones, iOS on 20.9% and Windows Phone on 3%. All these companies have design guidelines available for people that design applications for their mobile devices. These guidelines describe in detail what the application interfaces and icons should look like, but also how the application should behave. They aim to provide users of the platforms from the companies mentioned before with a consistent user experience among applications.

An application deployed to multiple platforms should offer the same user experience as well. Whether an Android, iPhone or desktop version of an application is used, users do not want to spend too much time learning the same application on another platform [36]. A good example is the Dutch train planner application *NS Reisplanner Xtra*, as it can be seen in Figure 3.1.

This presents challenges in interface design. Designers have to use the same interface across different platforms, while taking into account the guidelines for each of those platforms.

Context-awareness. Context-aware applications save user effort and frustration, and therefore increase the usability of mobile applications [22]. Exploiting contextual information may avoid manual user input at all [42], but in contrast also introduces new usability risks and challenges [27]. Research in this area fortunately resulted in guidelines related to context-aware applications. Examples include *always allow the user to take over control* and *avoid information overflow* [27]. Context-awareness should support users during their activities and not force itself to the foreground.

Sharing one's location with others is known to raise privacy concerns [49]. Therefore, for people that use a mobile device professionally, context-awareness most certainly can be a useful addition when applied correctly [22, 27].

Multimodal input. Multimodal interaction allows for easier input by using for example speech, scanning RF-ID tags, barcodes, passports, and can help by using GPS

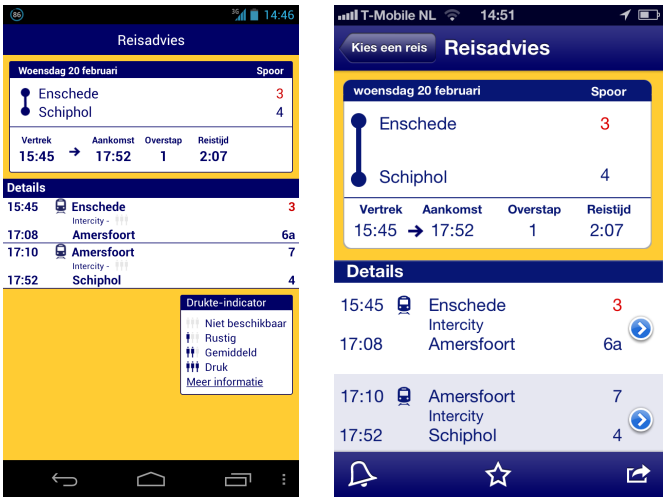

FIGURE 3.1: The *NS Reisplanner Xtra* train planner application on an Android smartphone (left) and iPhone (right). The applications show many similarities, but also platform specific characteristics.

or Bluetooth sensors [42]. Using multimodal input, specific types of input can be entered easier. For example, the journey planner shown in Figure 3.1 uses GPS to show nearby train stations to help select the departure station.

3.5.2 Low-level guidelines

While high-level guidelines describe in abstract terms how applications should behave, *low-level guidelines* describe in more detail how (parts of) an interface should be designed, or behave.

Design Patterns. Design patterns address specific issues in interface design [42]. As we mentioned earlier, text and data input in general is an issue on mobile devices. Many patterns exist on how to deal with specific issues in mobile interface design. An example is coping with text or other data entry. Guidelines prescribe that one should either use contextual information, auto-complete what the user is typing, allow the user to select some text from a list of predefined values, or use a specific input mechanism.

Building Blocks. The design guidelines by Apple [2], Google [23, 24] and Microsoft [38, 39] describe UI components such as text fields, dialogs, lists, grids, buttons, checkboxes and so on, which are available for designers and developers. The available interface components are frequently used user interaction components. These components help to provide a consistent user experience among applications on the same platform, as their behavior and layout is comparable. Because users recognize the components, there is no need to learn a new interaction technique, which increases the user experience.

Icons instead of text. Schröder and Ziefle conducted research towards a completely icon-based menu for mobile devices [50]. The textual terms used to indicate the function of a menu item is often ambiguous, difficult to comprehend and space

consuming — which does not help in coping with small screens. The use of icons in menus is encouraged: they are universal and improve the will to learn.

However, there are also downsides. Icons lack grammar, which is an issue because in language grammar is used to express relations between items. For this reason, Schröder and Ziefle argue that icons are not useful in hierarchical menus. In single-level menus, however, icons can be useful.

3.6 Success factors

To identify whether a developed application can be considered successful, we must first know how we can measure success. From literature we identified that *usability*, *fit for mobile work context* and *positive impact on work productivity* are key factors for the success of mobile applications in professional environments [56, 58].

Usability. Usability plays an important role in the acceptance of mobile devices and applications. In [5], Balagtas mentions that some usability aspects can be easily tested in a lab setting by having users interact with different versions of a user interface. *Usability*, however, is not a simple single property of a user interface, but has multiple components and is associated with five attributes, according to Nielsen et al. [40]: (1) *learnability* – the system is easy to learn, (2) *efficiency* – once the user learned the system, a high level of productivity is possible, (3) *memorability* – the system is easy to remember, so that the user can return to the system after a period of time not using it without having to learn the system again, (4) *errors* – the system should have a low error rate, and (5) *satisfaction* – the system should be pleasant to use and be liked by its users.

In particular, a positive correlation between perceived usefulness, fluent application navigation and a positive user experience was found in research [58]. These results suggest that satisfaction in navigational capacities as well as a central facet of easiness to use seem to be key variables in usability.

Fit for mobile context. Most usability measures are developed for desktop applications. For mobile applications it is important that the context in which an application is used is included. The interaction between the user and the application may fail if designers do not fully understand the context the device is used [56]. Users of mobile applications face more distractions than users of desktop applications.

In order to properly perform location dependent jobs, mobile workers need to know not only their location, but also the location of other parties or objects they either work for or have some kind of interaction with. Dynamic location-related information is critical for location-sensitive tasks [58].

Positive impact on productivity For companies adopting mobile technology, improving work productivity is usually considered to be one of the main success factors

from the company's point of view [56]. There are several benefits for the user of mobile applications in the field, of which one is time-saving by immediately registering factual data on-site.

3.7 Validation methods

A good user experience is important for users to adopt some technology [5, 17]. Evaluation and validation is integral to the design process. The goal is to improve the design, and thus the user experience [49]. Since a good user experience is good for business [17, 56], evaluations are an important part of the design process.

Because user experience is for a large part determined by the user interface, and we aim to design user interfaces based on guidelines, we believe that design guidelines can be evaluated by assessing the success of the mobile application.

Evaluations usually involve observing the end-user and measuring performance, in order to meet the exact needs of the end-user or customer [49]. Conducting evaluations for a finished product, which is called *summative evaluations*, is useful in order to assess the success of the product. Indeed, positive correlations were found between a positive user experience and an increase in technology usage [17].

In research we found that there are three types of evaluation that are useful during the evaluation of mobile device applications. We discuss each of them in more detail below:

Laboratory Studies. Being the used technique in 71% of all conventional usability evaluation tests, lab studies are the most used usability evaluation technique [10]. Several researches claim that 5 participants reveal about 80% of all usability problems, whereas 10 participants reveal approximately 90% [10]. This suggests one would need a relative small amount of participants in order to provide a good usability for most end-users. However, this argument is contradicted by others [16, 52]. Faulkner claims that one would need at least 10 participants in order to reveal a minimum of 80% of all usability issues [16].

Lab studies are used when design choices have to be made for a new product [49], or when usability tests must be conducted [10]. Often a series of tasks that are assessed have to be performed by the user. Questionnaires are used to register subjective responses, often in combination with (semi-)structured interviews.

However, lab studies cannot account for uncontrollable or external factors that play a role for mobile applications.

Field Studies. The advantage of field studies is that they show usability issues that may not be revealed during lab studies because of the external factors associated with the environment [10]. Especially for mobile applications, field studies are becoming more popular because of this advantage. They should be an integral part of the design process [49, 54].

Field studies are also used when the need for a new design exists, as they directly show issues encountered by users during the use of the application [49]. Comparable to how lab studies are conducted, during field studies one could also let users perform tasks, after which they would be asked to fill out a questionnaire asking for their experience.

Studies without users. Studies without involving users are conducted for two reasons. First, they are used in the beginning of the design process by asking consultants or research experts for their experiences or opinions [49]. Originally introduced by Nielsen, *heuristic evaluations* let experts assess whether UI elements conform to tried and tested principles [41, 49]. Additionally, *cognitive walkthroughs* are used to check whether the "user's goals and memory for actions can be assumed to lead to the next correct action" and involve simulating a user's problem-solving process [48, 49].

While applying each of these evaluations independently is useful to evaluate an application, combining these evaluation techniques may add significant value. Investigating from different perspectives is called *triangulation* [49]. The most applied form of triangulation is *methodological triangulation*: employing different data gathering techniques. Triangulation is good practice, but difficult to achieve. Data gathered from different perspectives may not be compatible, making it difficult to make the results complementary.

Chapter 4

Developing user interface design guidelines

This chapter presents the user interface design guidelines that were developed. It also presents the activities that were conducted in order to gather data that was required for the development of these guidelines.

4.1 Data Gathering

Before we started to develop guidelines, current issues with mobile devices at Sigmax were identified. This section describes the different steps in data gathering that were conducted. Four different steps of data gathering techniques were conducted: a literature study, expert interviews, field studies and a lab study. For each data gathering technique we describe the methodology and discuss the results in this section.

4.1.1 Literature study

Extensive literature research was conducted on the subject of challenges and approaches in applying Human-Computer Interaction concepts on mobile devices. The results of these studies have already been presented in Chapter 3.

4.1.2 Expert interviews

To get a better understanding of the current situation of mobile products by Sigmax that are already deployed in the field, such as user frustration and satisfaction, as well as limitations of products, expert interviews were conducted. The goal of the expert interviews is exploratory.

Methodology. The interviews were designed to be semi-structured, and each interview took 30-45 minutes. Questions asked were, amongst others, "*for which tasks in the professional environment should the mobile device offer support*", "*in which areas does the application support the user during his work?*" and "*in which areas does the application disturb the user during his work?*". The complete list of questions is included in Appendix A. The participants of the interviews were people that have contact with customers and end-users, because those people have a better understanding of how the products are experienced in the field when compared to software programmers. Five Sigmax employees were interviewed: two helpdesk employees, a sales manager, a functional consultant and the chief executive officer. Audio recordings and manual notes were made.

Open coding in grounded theory [9] was used to analyze the results. Open coding is an interpretive process meant to analytically break data into data units. In our case, the data was the interview and the data units were expressions by the participants. The analyst conducting open coding applies characterizing labels to data units. Related data units are labeled the same. Open coding is a process of questioning and constant comparison, helping analysts to break through subjectivity and bias. Its purpose is to identify the core topics in the data.

Results. Table 4.1 shows the labels that rose from the expert interview after open coding. The areas in which mobile applications are used differ. They are used by security companies in relatively device-friendly environments, but also by rail inspection companies or drainage cleaners where the environment is less device-friendly. With device-friendly we mean the likelihood of the device being damaged.

According to Table 4.1, software (SW ISSUE) and hardware issues (HW ISSUE) seem to be primary factors that influence user experience. One of the most heard comments is that users find that applications provide too little feedback about their activities. The applications tend to provide too little feedback when it comes to progress or user interaction, possibly providing a slow (SLOW) user experience. Changes in the user interface or application behavior after an update is reported to be confusing for users. Additionally, the design philosophy at Sigmax is that applications should be fully controllable with a finger, and without a stylus.

A hardware issue (HW ISSUE) is that people often find the quality of the device below what they expected. The quality here concerns the look of the device itself, the quality of the screen as well as the quality of the pictures made with the camera. While this may be true, a trade-off must be made. The devices that are deployed in the field are often *ruggedized*, meaning that the device is physically strengthened to better resist excessive use or abuse. However, this also means trade-offs have to be made considering the quality of the device. It is also clear that the mobile application could be more supportive in terms of feedback.

A mobile device forces the user to work in a certain way, which is a point of attention (PROC POA). While this certainly provides benefits (PROC PRO), especially for new users who have to learn working with the device, it may be less desired by experienced workers.

Inputting information is found the be difficult by only a few (INPUT ISSUE). The use of multimodal input is seen as a benefit (MULTIMODAL PRO). Certain hardware capabilities (HW PRO) allow for multimodal input by, for example, using GPS sensors.

Label	Description	#
HELPS	Data units with this label indicate that people are pleased with a mobile device that helps them during their jobs.	10
SW ISSUE	Data units with this label indicate that people experience software issues.	5
PROC POA	Data units with this label indicate that it is a point of attention that a mobile device forces a certain way of working.	4
HW ISSUE	Data units with this label indicate that people experience hardware issues.	3
HW PRO	Data units with this label indicate that people gain advantages by using mobile device hardware and its capabilities.	3
PRIV ISSUE	Data units with this label indicate that people have the feeling their privacy may be infringed.	2
INPUT ISSUE	Data units with this label indicate that people have issues inputting data.	2
MULTIMODAL PRO	Data units with this label indicate that the multimodal input (i.e., using device capabilities to help the user during input) is experienced to be pleasant.	2
PRES	Data units with this label indicate issues with the presentation of data on-screen.	1
SLOW	Data units with this label indicate that the device is experienced to be too slow.	1
PROC PRO	Data units with this label indicate that the mobile devices force a certain way of working, which is seen as a good thing.	1
SW FANCY	Data units with this label indicate that people are pleased with software features.	1

TABLE 4.1: Labels, including their description and number of occurrences, that were derived from the expert interviews by applying open coding in grounded theory.

Worth mentioning is that the top heard comment during the interviews is that users of mobile devices are very pleased with its use (HELPS). During a single interview we learned that screen animations were found to be very nice (SW FANCY).

4.1.3 Field studies

After the expert interviews, the next step in data gathering were field studies in the form of observations. Observations conducted in the beginning of the design process help understand the user context, tasks and goals [49]. This section provides an overview of the most important aspects of the methodology used for field studies and discusses the results. Appendix B contains a more detailed description of the methodology and detailed results.

Methodology. The field study was set up as observation in combination with informal interviews before, during and after the study. During the study, we adopted a passive attitude, so that application behavior by the user is interrupted as little as possible. The simple *who-what-where-framework*, as introduced by Rogers et al. [49], was adopted to keep the goals and targets straight. The observed person was asked to think aloud while interacting with the mobile device. Three field studies were conducted: at a company that cleans sewers and drainage systems, at

a railway inspection company, and at a municipality where parking attendants use a mobile device to register parking violations. Written notes were made, and after each observation the day was reflected upon by the researcher by writing down as many observations as possible. Open coding in grounded theory was used for analysis. The most important observations and claims by the observed persons served as data units.

Results. Table 4.2 shows the labels that rose from the field studies after open coding. From Table 4.2 we learn that most observations had to do with how the mobile application presented its data (PRES), the mobile device being slow (SLOW), hardware issues (HW ISSUE) and software issues (SW ISSUE).

During field studies we primarily discovered that people often find themselves browsing through long lists of items using a stylus, something that has to do with how the application presents it data. In one field study, the user had to scroll through lists so often that the protective screen cover was completely worn off at certain parts. In another field study we noticed that people always had to click on the single search result to view the details, instead of having the details immediately shown. Additionally, in some cases the device lacked human-understandable error messages. For example, a `NullPointerException` was referred to as a "*nul-error*" ("*nul*" being Dutch for zero), while it should not be shown at all.

We also learned a lot about how users interact with the device (INT POA), and how the application matches their work process (PROC POA). User interaction in all three studies was done using a stylus. This is an interesting observation, since during the experts interviews discussed in Section 4.1.2 we discovered that the design philosophy is that interaction should be possible using only fingers.

The users did not seem to have much problems with how the current interaction works (INT ISSUE), and are happy with the device being around and facilitating their jobs (HELPS).

An often heard comment was that the application is experienced to be slow, and keeps getting slower after each software update. The mobile application does not provide proper feedback about progress when time consuming algorithms are being executed, which most likely strengthens this experience. Scrolling through long lists of choices in order to select an option is experienced to be frustrating. The observed people argue that the application could place items that are often selected from some list on top of the list, so that scrolling takes less time.

4.1.4 Lab study

During the literature and field studies we found that structuring hierarchical information is difficult, and that it is best to only show 4-8 items per hierarchical level [1, 21, 28]. However, the mobile devices that were used in this research are by now outdated and replaced by smartphones and tablets. To understand how people prefer browsing hierarchical data structures on both smartphones and tablets, a lab study was conducted.

This section provides an overview of the most important aspects of the methodology and discusses the results. Appendix C contains a more detailed description of the methodology and detailed results.

Label	Description	#
PRES ISSUE	Data units with this label indicate issues with the presentation of data or information in the application being too vague or unclear.	8
SLOW	Data units with this label indicate that people experience the device to be slow.	5
HW ISSUE	Data units with this label indicate that people experienced hardware issues.	4
INT POA	Data units with this label indicate interaction with the device is a point of attention.	3
SW ISSUE	Data units with this label indicate that people experienced software issues.	3
PROC POA	Data units with this label indicate that the device helps during people's jobs, but their work process must be taken into account. A point of attention.	3
HELPS	Data units with this label indicate that people are pleased with a mobile device that helps them during their jobs.	2
INT ISSUE	Data units with this label indicate that people experienced issues with device interaction.	1
QUALITY ISSUE	Data units with this label indicate that people experienced quality issues.	1

TABLE 4.2: Labels, including their description and number of occurrences, that were derived from the field studies by applying open coding in grounded theory.

Methodology. A simple application was developed to navigate hierarchical data structures on smartphones and tablets. On the tablet, the application exploited the larger screen. Figure 4.1 and 4.2 show screenshots of the application on both a smartphone and tablet. Two data structures were implemented: one that shows a low amount of items on each hierarchical level but uses more levels (the *deep* structure), and one that used longer lists where scrolling through the list would be required, but less hierarchical levels (the *broad* structure). In this study there are two independent variables: smartphone versus tablet, and a deep versus broad data structure, providing us with four different configurations. This 2x2-study was designed as a counterbalanced within-design lab study using four different treatments. Participants were handed a form with instructions and tasks to conduct. Furthermore they were equipped with a smartphone and tablet to conduct the tasks, and a laptop computer to fill out a questionnaire about how they experienced each task.

The questionnaire was used to register user experience. A selection of questions that were asked in the questionnaire are: "*I had to perform too many steps before I got to the item I wanted*", and "*There were too many choices from which I had to choose at each step*". These questions were answered using a 7-point Likert scale where 1 meant "*I completely disagree*" and 7 meant "*I completely agree*". At the end of each single experiment, a short semi-structured interview was conducted to get a better understanding of the people's preference for the smartphone or the tablet, and the deep or the broad data structure. Additionally, video recordings were made to allow interaction behavior to be reviewed later.

A total of 16 people participated in the lab study (10 male, 6 female, average age 24.4 years, 9 academics). All participants owned a smartphone and used it daily. The intensity of use was measured on a 7-point Likert scale. Most participants use their smartphone intensively (*M* =

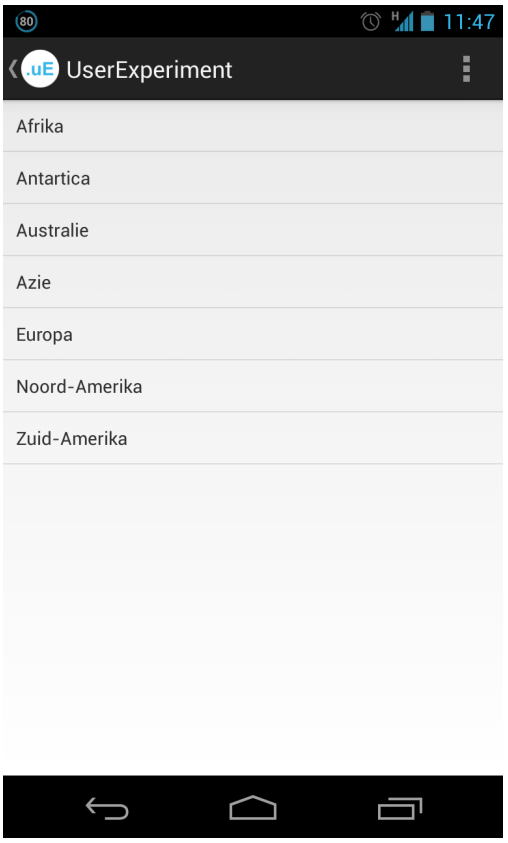

FIGURE 4.1: Screenshots of the application that was used during the lab study on a smartphone.

5.5, *SD* = 1.751). Tablets were owned by less participants: 25% of the participants owned a tablet, and only 19% of all participants used a tablet on a daily basis. This means some participants use a tablet on a daily basis, without owning one. Of the 4 participants that owned a tablet, there was only one person that uses it on a daily basis. The intensity of tablet usage was also much lower (*M* = 2.33, *SD* = 1.211) than that of smartphones.

Results. First of all, people found the assignments during the study very easy for each of the configurations: *M* = 5.94, *SD* = 1.569 for smartphones using the deep data structure, *M* = 5.88, *SD* = 1.996 for smartphones using the broad data structure, *M* = 6.25, *SD* = .856 for tablets using the deep data structure, and *M* = 6.25, *SD* = 1.291 for tablets using the broad data structure. No significant differences were found in how easy people found the deep versus broad data structure on a smartphone ($t(15) = .102$, $p = .92$) or tablets ($t(15) = 0$, $p = 1$). There were also no significant differences found in how easy people found the deep data structure on a smartphone versus tablets ($t(15) = -.924$, $p = .37$), or the broad data structure on a smartphone versus tablet ($t(15) = -.651$, $p = .525$). It is apparent that this is related to the fact that most people were already acquainted with smartphones, and some with tablets.

Paired T-tests showed two significant comparisons regarding the extent to which users had to go through too many hierarchical levels using one of the data structures. This user experience was measured on a 7-point scale, by asking participants if they had to go through too many hierarchical levels to complete an assignment (1 = completely disagree, 7 = completely agree). On

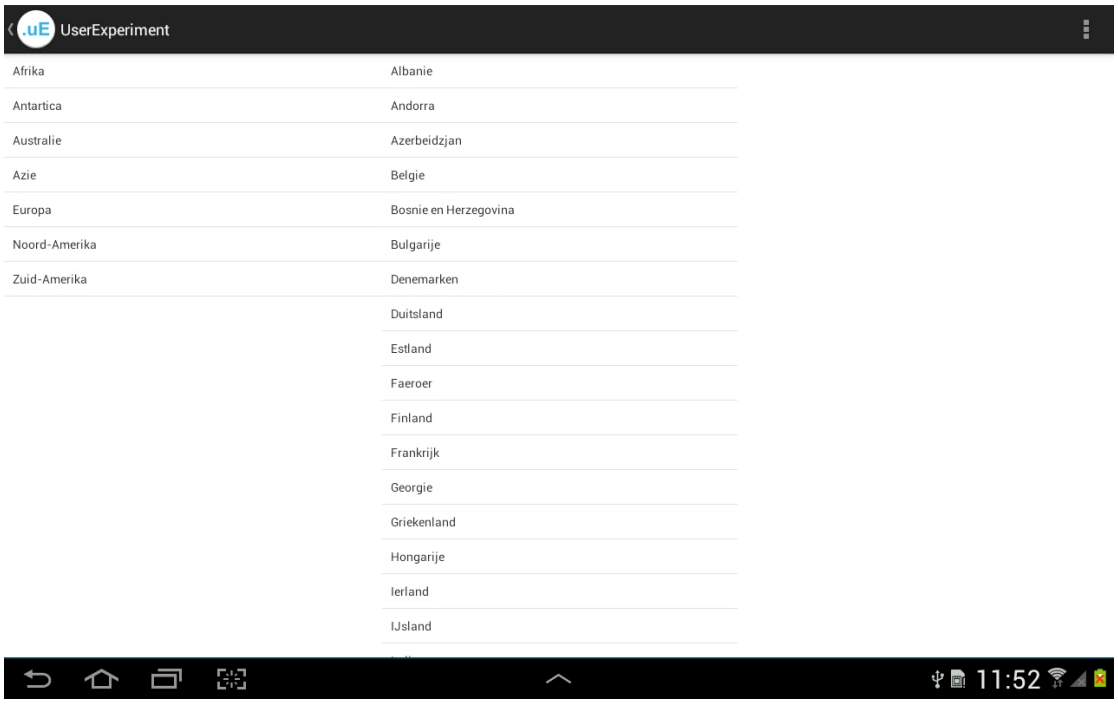

FIGURE 4.2: Screenshots of the application that was used during the lab study on a tablet.

smartphones, a significant difference ($t(15) = 3.135$, $p < .01$) was found between participants that found they had to go through more *too many hierarchical levels* using the deep data structure ($M = 3.88$, $SD = 2.217$) than using the broad data structure ($M = 2.69$, $SD = 1.702$). This comparison was also significant when participants had to use a tablet ($t(15) = 3$, $p < .01$). On the tablet, people also found that the deep data structure yielded more *too many hierarchical levels* ($M = 3.94$, $SD = 2.175$) than the broad data structure ($M = 2.81$, $SD = 1.797$).

The extent to which participants found that they had *too many choices at each step* (in the hierarchy), we see that on smartphones there is no significant difference ($t(15) = -1.291$, $p = .216$) using either a deep ($M = 2.63$, $SD = 1.708$) or broad data structure ($M = 3.38$, $SD = 1.668$). On tablets this difference was marginally significant ($t(15) = 1.962$, $p = .069$) using either a deep ($M = 3.25$, $SD = 1.77$) or broad data structure ($M = 2.38$, $SD = 1.408$). This suggests that participants are more bothered by the number of levels they have to go through and care less about the number of items at each level, provided that the list is properly sorted for its goal.

We also tried to find correlations between summations of data. The questionnaires were used to measure to what extend people had to perform too many actions to either select an item from the list, or get to the correct level in the hierarchy. We summed up these two questions for each of the configurations for all participants, constructing a new measure that indicates how much a participant did not like a certain configuration. The configurations are in this case all possibilities of the independent variables (smartphone versus tablet, and a deep versus broad data structure). Using this approach, we found one significant comparison between how participants experienced the deep versus the broad data structure on tablets ($t(15) = 2.928$, $p = .01$). The participants experienced the action of having to go through *too many hierarchical levels* and having *too much choices at each step* were summed both for the deep and broad data structures.

Participants were more pleased with a broad data structure (M = 5.19, SD = 2.762), than a deep data structure (M = 7.19, SD = 3.6).

These results suggest that people prefer longer lists if that means they have less hierarchical levels. The interviews confirm this: participants mentioned that they do not experience scrolling as a bad thing, since people that use mobile devices “are already used to it". These results are in contrast with earlier research that was discussed in Section 3.3. In that work, it was suggested that 4-8 items per hierarchical level would be optimal [21]. However, the comparison with this study is not entirely fair, as our experiment was conducted with modern smartphones and tablets, while Geven et al. conducted their experiments in 2006 using devices with small screens and limited interaction mechanisms.

It is safe to say that when displaying hierarchical data structures, people prefer longer, logically sorted, lists at each level if that means they have to go through less hierarchical levels.

4.2 Design Guidelines

This section presents the guidelines that were obtained from the data gathering techniques. Similar to what we found during the literature study presented in Chapter 3, a distinction is made between high and low-level design guidelines. The distinction between high-level and low-level guidelines was made to categorize the design guidelines based on their purpose.

The guidelines were defined based on what we learned during data gathering. Issues that occurred often, and in multiple steps during data gathering formed the basis for these design guidelines.

During the literature study, it was found that three success factors are of importance for mobile applications: *usability*, *fit for mobile work context* and *positive impact on work productivity*. The aim of the guidelines is to help achieve one or more of these success factors.

4.2.1 High-level guidelines

High-level guidelines can be seen as principles that should be taken into account during the development process of user interfaces, as well as the application itself.

Be responsive

Identified from: expert interviews, field studies.

Motivation: An often heard comment during the expert interviews and field studies was that the application would become slow or irresponsive. If an application does not respond immediately to user action, this may result in frustrated users as their work is slowed down and their actions are not immediately acknowledged. Users might also think that the application crashed.

Desired behavior: Regardless of whether it is possible to immediately provide a proper response to user action, the application should always provide on-screen feedback when a user

interacts with the device, albeit a loading or 'please wait' dialog. This way the user knows that the device registered the action and is not tempted to try again. If an action is being executed that takes some time to finish, there are three degrees of information that can be displayed: (1) a progress dialog that shows how long the operation is going to take, (2) a percentage that shows the current progress, (3) a 'please wait' dialog.

Be consistent

Identified from: literature study, expert interviews.

Motivation: After a while, users will get acquainted with the application and expect certain application behavior. During the field studies we learned that people sometimes get confused after an application update, so it is important to keep this in mind. Consistency decreases the learning curve: once people learned a certain concept in an application, they recognize it when the concept is re-used elsewhere in the application.

Desired behavior: To *be consistent* is a guideline that effects several dimensions, such as types of screens, but also how on-screen elements respond to user action. The same types of screens and concepts should be used throughout the application. Components that are often used should be placed in a framework or automatically generated, so that re-use is encouraged. Because mobile devices enforce a way of working for the user, it is important to offer consistency so that people recognize all screens and do not have to learn a new concept every time.

Provide understandable feedback

Identified from: expert interviews, field studies.

Motivation: While developers are technical, most users are not. When a user clicks a button on the screen but nothing happens, or when something went wrong and a technical error message is displayed, it worsens the user experience and people may develop an aversion to the application. In case of application error, people tend to think they did something wrong while this may not be the case. Feedback is important to keep the user informed about the activities of the application, in addition to being responsive.

Desired behavior: At all times, but especially in case of errors, it is of utmost importance that an application provides understandable feedback. User friendly error messages should be displayed, including an error code that allows the user to quickly look up the error to inform the helpdesk or developers. When possible, the reason why the error message popped up should be included. In case the user made a mistake, the user should be informed of the mistake and asked to try again. In these cases, an error code does not have to be displayed, since the error can be fixed by the user.

Be supportive and minimize manual input

Identified from: literature study, expert interviews, field studies.

Motivation: Manual input is experienced by users to be difficult and time consuming. Reasons include a small screen — making an on-screen keyboard tiny and difficult to use, especially on smartphones — as well as the limited input capabilities of mobile devices in general. Therefore, manual input should be reduced and the application should support the user as much as possible.

Desired behavior: An application should avoid user input and use additional device capabilities such as GPS sensors, or the ability to scan barcodes in order to support the user in his or her tasks, as explained in Section 3.3. Developers should think about tailoring to specific types of users; different users using the same application may have different preferences or activities. *To be supportive* is a paradigm that should be followed through the whole process of designing and implementing of an application.

Prioritize data presentation in list choices

Identified from: field studies.

Motivation: Users often have to make a choice from an extensive list of options. The organization of this list is important, as extensive lists are prone to claim lots of user time. In some cases it may be a challenge to show an extensive list of items, as the screen size of mobile devices is smaller than that of desktop or laptop computers. This guidelines affects list choices used to select a single option, and not a full-screen list with items.

Desired behavior: The desired behavior for this guideline is strongly related to being supportive. The information needed by the users should be displayed immediately, and not after an extra (unnecessary) user action. Additionally, users should not have to spend too much time choosing an option from a list of possible choices. In case a list of options is displayed, the option that is most likely to be selected should be presented at the top of the list.

Allow customization

Identified from: literature study (Section 3.5.1).

Motivation: Consultants and programmers will often determine what should be displayed in a list. However, users may want to be able to customize lists in order to apply a certain filter to reduce the number of items in a list and find specific items. If the application supports a home-screen, it may differ which items are more often selected by certain users. Allowing customizations makes users feel comfortable and in control.

Desired behavior: Developers should carefully consider what should be customizable. For example, allowing to customize filters that are applied to certain lists, the displayed data can be tailored to the users needs at that moment. Other customizable aspects could be a home screen where the items that need to be displayed can be selected.

4.2.2 Low-level guidelines

Low-level guidelines provide specific instructions in how to implement application behavior for certain concepts. These guidelines go into a lot more detail than high-level guidelines. While a significant number of high-level guidelines were developed, only one low-level guideline was developed. The reason for this is because our studies revealed more principles that should be taken into account during development than concrete advises on how to build a user interface.

Navigating through hierarchical data structures

Rose from: literature study, lab study.

Motivation: When displaying hierarchical data structures, a trade-off between the number of hierarchical levels and number of items per level has to be made. It may be difficult to make this trade-off and anticipate on how users are going to experience navigating through hierarchical structures.

Guidelines: The most important guideline is that the items in a list should be sorted logically. In most cases this will be alphabetically, but developers must take into account the goal of the list. For example, if a list of articles is displayed with a price, it may very well be that the most expensive items should be on top. Additionally, people do not mind scrolling through longer lists, as long as the items are logically sorted and therefore easy to find. It is useful to add a search box that allows textual input on top of the list that acts as a filter for the list of items when text is entered.

Smartphone implementation: Lists use full screen width and height, and should be headed by a search box. The search box allows to construct a filter for the current list based on textual input. In case a list is contained within some other screen, the list uses the full screen width, and is as high as is required to show all items in the list. In this case, a search box is not required, but the items should be sorted logically.

Tablet implementation: How lists are shown depends on their usage. In case details are shown when an item is clicked, the list uses about a third of the available screen width and the complete available height. The width should be about a third, but should be kept the same for different list screens throughout the application, as suggested by the guideline to *be consistent*. The selected item remains highlighted. When a screen is shown in a pop-up, it uses the complete pop-up width and height. If a list is used in some other screen, it is as wide as the screen, and as high as is required to show all items. A search box is not required, but the items should be sorted logically.

4.3 Discussion

The development of design guidelines involved a lot of data gathering, in which many opinions have been registered. All these opinions were categorized and some of them were shared by more people, without these people knowing each other. Especially opinions that were registered more than once and from different data gathering techniques formed the basis for our

design guidelines. The primary opinions concerned frustrations about general behavior of the device, which is why there are more high-level guidelines, or principles, than low-level guidelines being presented.

Chapter 5

Solution design

This chapter presents the solution design of a Model-Driven Engineering environment that supports the development of multi-target mobile applications. We present the metamodels and model transformations that were defined in this work.

5.1 Introduction

One of the goals of this research is to develop a proof of concept environment that supports Model-Driven Development of mobile applications for both smartphones and tablets, and in particular with respect to mobile application layouts. To implement an MDE environment there are three key requirements that need to be satisfied, being (1) a language to support the development of metamodels and models; (2) a language to support model-to-model transformations; and (3) a language to support model-to-text transformations.

The language for defining metamodels was chosen to be *Ecore* [13], a popular modeling language in the academic world. For model-to-model transformations the *ATLAS Transformation Language* [30, 31] (ATL) was adopted. For model-to-text transformations the *Xpand* [14] language was adopted. Conveniently, the *Eclipse Modeling Project* [15] bundles tools that support these three languages, and provides these languages as plugins for *Eclipse* [12], a popular integrated development environment (IDE) in both the academics and industry.

The user interface design guidelines and principles that were presented in the previous chapter, were used as input during the development of metamodels and transformations. This combination could show the real added value of using MDE in the development of multi-target mobile applications.

To keep the proof of concept feasible and within time boundaries, we limit ourselves to the development of a Model-Driven Engineering environment that generates an application for the Android platform.

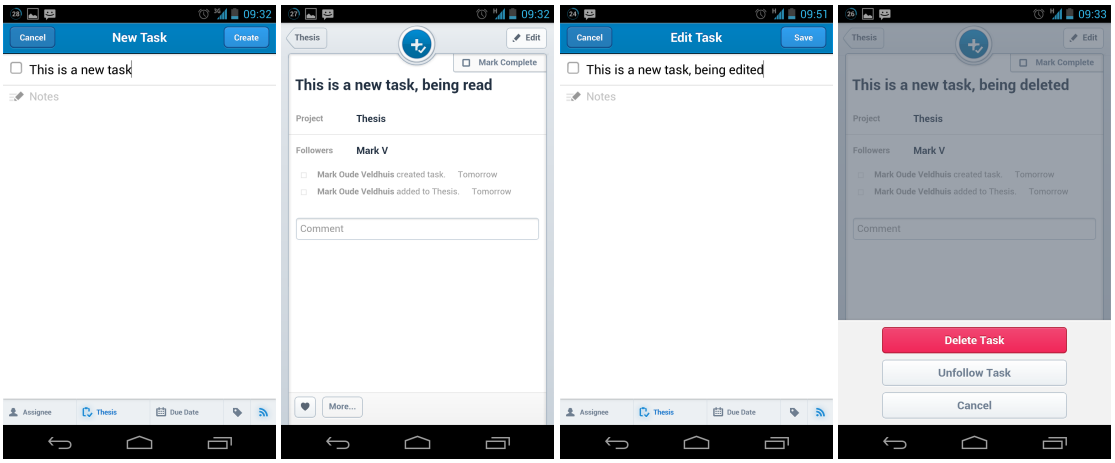

FIGURE 5.1: Examples of screens involved with a classic CRUD-pattern for an entity, in this case tasks. From left to right: creating, reading (viewing details), updating and deleting. The application shown is *Asana* on an Android smartphone.

In the remainder of this chapter we start by discussing the philosophy that was adopted for the design of applications in an Model-Driven Engineering environment. Then, the metamodels that were developed are presented and explained in detail, followed by the model-to-model and model-to-text transformations. We end this chapter with a discussion.

5.2 Application patterns

From the analysis of the current mobile applications at Sigmax we learned that there are several abstract *application patterns* used throughout applications. An application pattern can be seen as a building block that offers functionality. An instance of a pattern requires some configuration, so that it provides the desired functionality.

For our proof of concept implementation, we adopted a single pattern: the *CRUD*-pattern, which is used to **c**reate, **r**ead, **u**pdate and **d**elete instances of some entity. You can find this pattern in many applications today. Figure 5.1 illustrates an example CRUD-pattern for a task management application on a smartphone. It shows a list of your tasks, where you can also create new tasks, update existing ones, or delete tasks that are no longer needed.

Application patterns can be broken up into different types of screens, each with their own specific goal. Most data-driven applications only have a few different types of screens that are differently implemented to satisfy requirements. Following the CRUD application pattern that was selected for the proof of concept, we identified three different types of screens: (1) A *dashboard* menu to navigate to other screens; (2) a *list screen*, used to display a list of instances of some entity; and (3) a *details screen* that shows details of a single entity instance.

The CRUD-pattern that we described as an application pattern can be implemented using these types of screens. A list screen may be used to show an overview of entity instances. A details screen can be used to show details of a single entity, allowing this entity to be updated, or to create a new entity. Figure 5.1a and 5.1c shows that the screens to create or update a task are identical.

5.3 Metamodels

We developed three metamodels: two to allow the definition PIMs, and one to allow the definition of PSMs. Introducing multiple steps in metamodeling allows us to specify specific characteristics or behavior at the most suitable levels. It also decreases the complexity of the model transformations, as they are being split up over multiple models. The metamodels define what elements are available to create specific models, and what relations may be defined between the model elements.

The three metamodels that were developed are: (1) the *Sigmax*-metamodel, allowing to model applications in terms of application patterns; (2) the *Screens*-metamodel, allowing to model an application in terms of types of screens; and (3) the *Android*-metamodel, allowing to model an application for the Android platform. Multiple metamodels were defined to distribute the complexity of application development over multiple models. At the most abstract level, the Sigmax-metamodel hides as much implementation details as possible.

The metamodels are written in ECore [15], which looks similar to UML. Many of the modeling elements in ECore correspond to those of UML. It recognizes elements such as `EPackage`, `EClass`, `EOperation`, `EAttribute`, `EReference`, amongst others.

First, the **SigmaxApp**-metamodel was developed to facilitate the development of a model that represents a mobile application in terms of application patterns that are relevant to the domain, in our case Sigmax. It supports the development of PIMs, and aims to keep the design of an application as simple as possible.

Second, the **Screens**-metamodel was developed to represent the application under development in terms of logical screens and their relations. The goal of this metamodel is to allow the development of PIMs that break down the application patterns from the SigmaxApp-metamodel into different types of screens that are related to each other. A bit of logic is introduced in this model, allowing to infer application behavior to differentiate the purpose of screens.

Third, **Android**-metamodel was developed to support the development of PSMs that represent the application on an Android platform. It supports modeling application behavior so that logic can be added during code generation, as well as multiple layout definitions for a single screen. This allows us to create a PSM of the application under development that can be deployed on multiple devices with different screen sizes, following the development paradigms of the Android platform.

5.3.1 *SigmaxApp*-metamodel

The SigmaxApp-metamodel is the most abstract metamodel. It facilitates the development of PIMs, and its goal is to keep the design of the application as simple as possible. It does this by providing model elements that represent application patterns that are often seen in applications, bound to a certain domain or context of use. The number of details required to create a model based the SigmaxApp-metamodel is kept to a minimum. Application patterns have a set of properties that need to be set in order for them to offer the desired behavior and features.

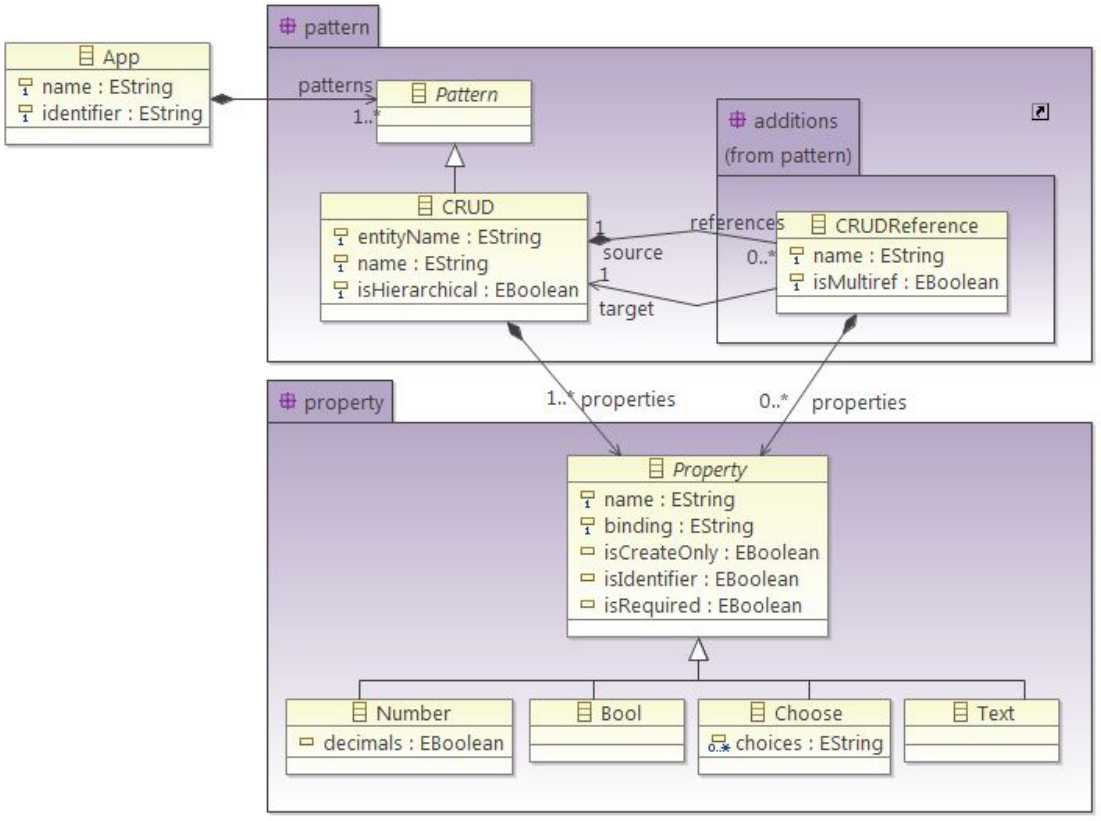

FIGURE 5.2: *SigmaxApp*-metamodel.

Figure 5.2 shows the SigmaxApp-metamodel. As can be seen in this figure, the EClass `Pattern` is an abstract, attribute-free concept. It was introduced so that the metamodel can introduce multiple specific application patterns, such as the *CRUD* pattern. We discuss this pattern in more detail below.

The `CRUD` EClass is added to a model in order to represent a CRUD-pattern for some entity, allowing instances of that entity to be created, read, updated and deleted. Using the `name` attribute, the application pattern is provided with an identifying name such as "*Location Management*" if the pattern is meant to create, read, update and delete locations. The attribute `entityName` is required so that data from the correct data source can be retrieved and manipulated. The attribute `isHierarchical` tells whether an entity is hierarchical, as that requires a specific way of user interaction, determined by the user interface guidelines that were discussed earlier in Section 4.2. Furthermore, a CRUD-pattern recognizes multiple properties. These properties have a certain data type, and specify which properties of the entity can be modified. A `CRUDReference` is used to relate entities, and can be enriched with properties.

The SigmaxApp-metamodel represents few user interface specific details. It does not allow to specify how a certain pattern should be implemented on a smartphone or on a tablet. The reason for this is because this metamodel should be as abstract as possible. One of the important guidelines of user interface design is to be consistent. Allowing platform-specific specifications at this level of modeling could jeopardize this guideline.

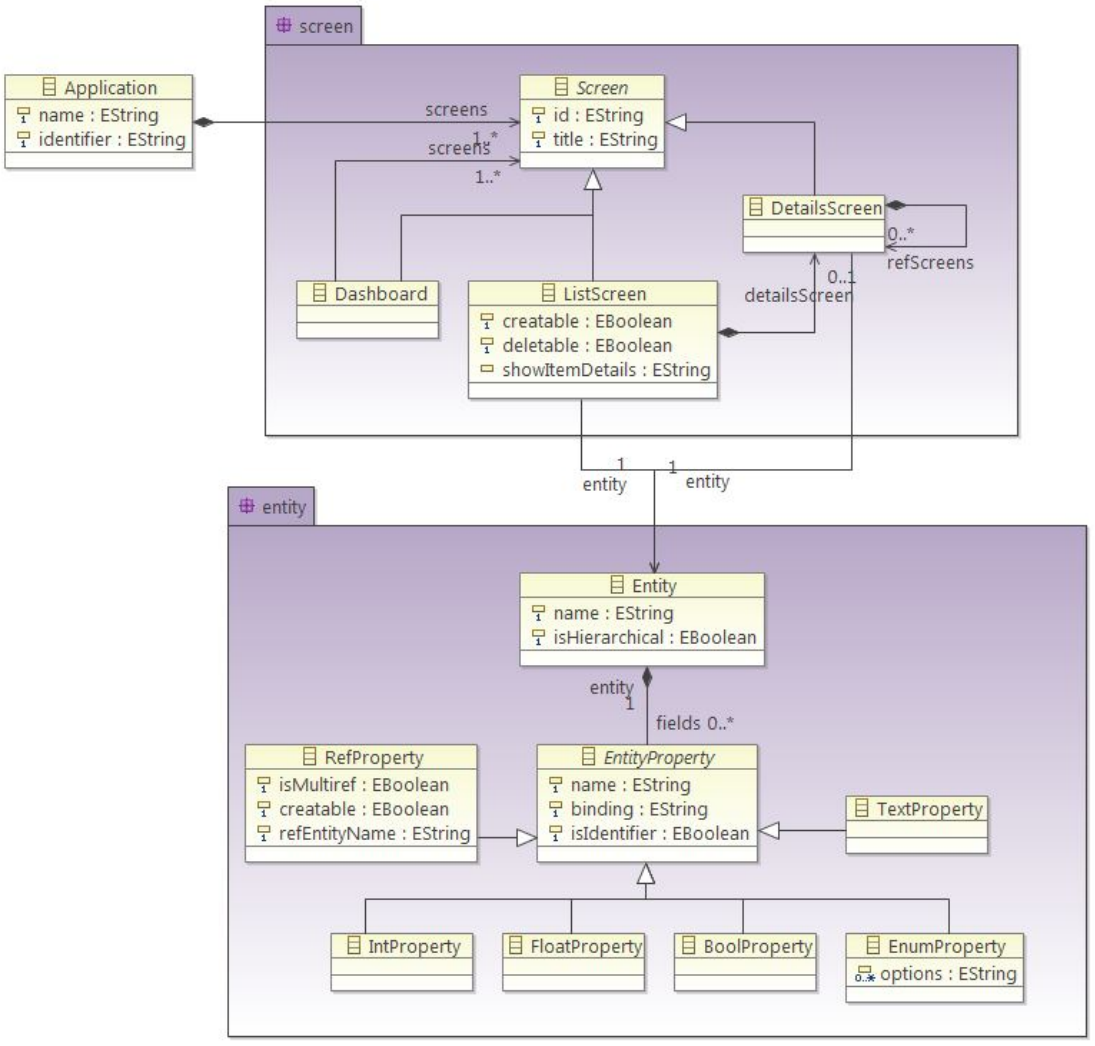

FIGURE 5.3: *Screens*-metamodel.

5.3.2 *Screens*-metamodel

The goal of the Screens-metamodel is to support the creation of PIMs of the applications under development in terms of screens and their relations. Figure 5.3 shows the Screens-metamodel. The goal of this model is to advance a step in application development. It does this by breaking down the application patterns from the SigmaxApp-metamodel into different types of screens. In other words, it further specifies what is visible to the end-user of the application in terms of screens.

The metamodel recognizes different types of screens that may each have their own additional properties. The types of screens that are available represent the goals of application patterns. The Screens-metamodel recognizes three types of screens: (1) the `ListScreen`, meant to list instances of a certain entity; (2) a `DetailsScreen`, meant to view the details of a single entity or update its information; and (3) a `Dashboard`, meant to navigate to other screens.

The Screens-metamodel does not provide a method to specify application behavior or logic, as this is implicitly defined. For example, a ListScreen may have a reference to a DetailsScreen, indicating that navigation to a DetailsScreen from a `ListScreen` is possible, but the metamodel does not specify how the `DetailsScreen` can be reached from the `ListScreen`. Instead, the reference between these types of screens implies that navigation to the `DetailScreen` is possible by selecting an item in the `ListScreen`. How this selection is implemented is not specified in this model, as this may depend on the target platform and therefor be specified in a succeeding model. Additionally, referring to an `Entity` from a `ListScreen` implies that some data storage must be consulted to obtain a list of instances from that entity.

5.3.3 *Android*-metamodel

The Android-metamodel facilitates the creation of PSMs. Its goal is to support the creation of models that represent applications for an Android platform. Figure 5.4 shows (part of) the Android metamodel. The available modeling elements in the Android-metamodel represent the development components on Android, supplemented with modeling elements that allow to model logic and application behavior.

An `Activity` represents a screen on an Android device and uses no, one or multiple `Fragments`. Android introduced fragments primarily to support flexible user interfaces on larger devices such as tablets, without having to develop a second version of an application. Fragments represent certain behavior or a portion of the user interface within an activity, and thus allow reuse. The Android development platform provides extensive support for the development of multi-target applications. This metamodel was developed with this support in mind. A direct result from this can be observed in Figure 5.4: both an `Activity` and `DialogFragment` recognize one or multiple layout definition. A `Layout` has one attribute, namely: `target`. This string identifies for which screen size the layout is suited, as supported by the Android platform. The Android metamodel aims to provide the required modeling elements to support the creation of a single model for an Android-application, aimed at both smartphones and tablets.

Furthermore we introduced modeling components that can enrich a model with logic. An example is an `Activity`, which is an abstract concept in our metamodel. A specific implementation is the `DashboardActivity`, which is used to start other activities. A `CRUDActivity` was introduced to share attributes and references that the `ListActivity` and `CreateActivity` have in common. Certain `ActivityBehavior` can be attached to an activity, to allow dynamic configuration of a certain activity instance.

5.4 Transformations

The development of multiple metamodels requires multiple model transformations. Since three metamodels were developed, two model-to-model transformations are required. Additionally, a model-to-text transformation is required to generate the required source code for an application that can be run on an Android device.

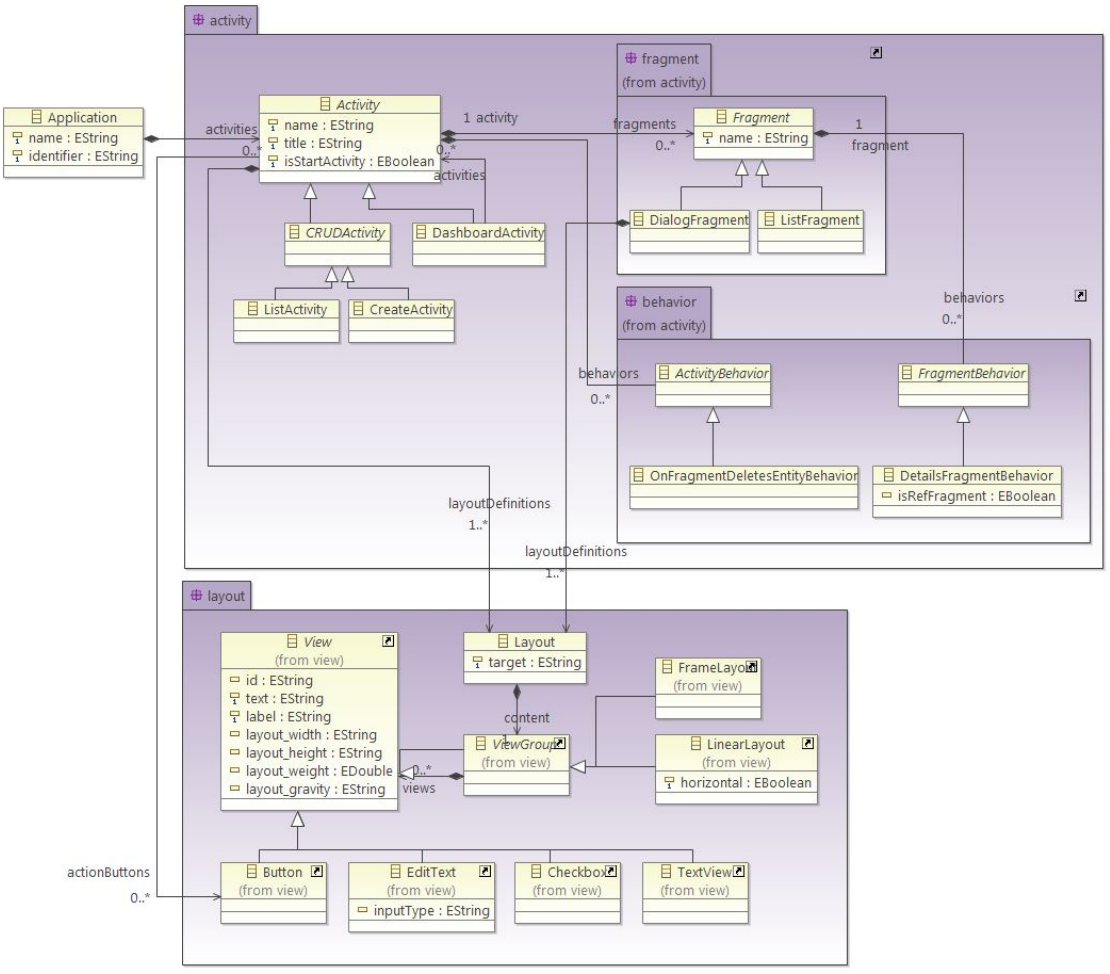

FIGURE 5.4: Excerpt of *Android*-metamodel. This figure only illustrates the most important parts of the Android metamodel.

The model-to-model transformations were implemented in the declarative ATLAS Transformation Language (ATL). The model-to-text transformations were implemented in the imperative language Xpand. Both languages are available within the Eclipse Modeling Project (EMP) [15] .

Figure 5.5 illustrates the model-to-transformations that were introduced, including their operational context. The green elements are components of MDA; the yellow elements were metamodels and transformations written during this work; the blue model is made by developers using the MDE environment; and the red elements are models derived through transformations The model-to-model transformations are discussed in further details in this chapter. Figure 5.5 omits the model-to-text transformation that generates source code from the Android model $M_{Android}$.

5.4.1 Model-to-model transformations

Two model-to-model transformations were introduced. The first transforms a model instance of the the SigmaxApp-model, $M_{SigmaxApp}$, to a model instance of the Screens-metamodel, $M_{Screens}$. In Figure 5.5 this transformation is referred to as $T_{SigmaxApp2Screens}$. The second

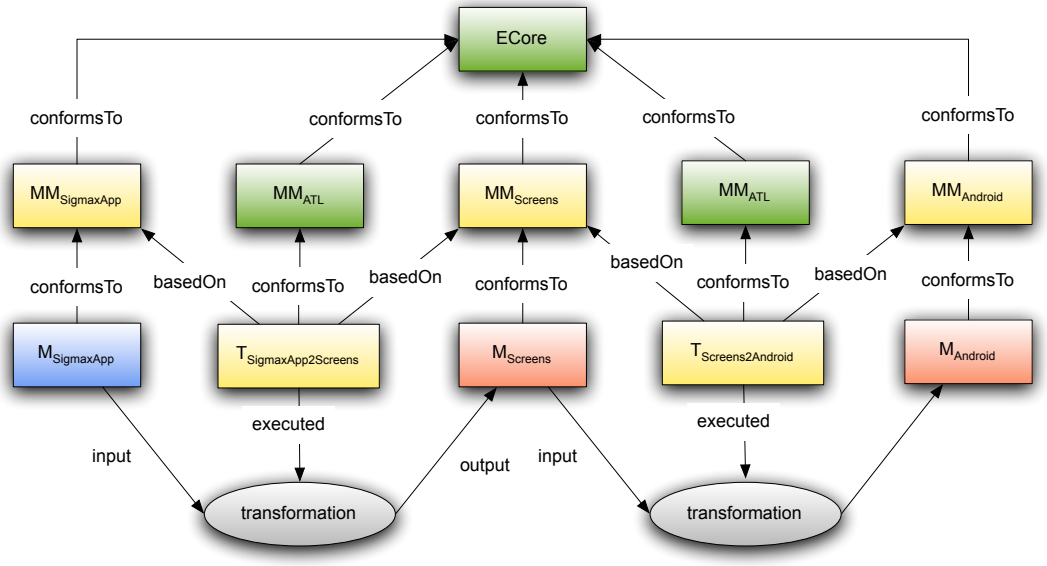

FIGURE 5.5: The model-to-model transformation chain and its operational context.

Source element	Target element
App	Application Dashboard
CRUD	ListScreen DetailsScreen Entity
CRUDReference	DetailsScreen Entity RefProperty (2x)
...	...

TABLE 5.1: Transformation rules of how $T_{SigmaxApp2Screens}$ transforms source elements from the SigmaxApp-metamodel to target elements for the Screens-metamodel. This Table does not include the mappings for the entity properties, they are related one-to-one.

transformation transforms the generated $M_{Screens}$ to a model based on the Android-metamodel, $M_{Android}$. This transformation is referred to as $T_{Screens2Android}$. We describe each of these transformations in more detail below.

SigmaxApp to Screens. The $T_{SigmaxApp2Screens}$ transformation is the first step in the transformation process. From the SigmaxApp-model that was manually created, it derives a second model that is based on the Screens-metamodel. We call this the Screens-model, or $M_{Screens}$ as it is referred to in Figure 5.5.

Recall the SigmaxApp and Screens metamodel from Figures 5.2 and 5.3, respectively. The simple, manually created, SigmaxApp-model must be transformed to a more detailed model. Table 5.1 shows how the SigmaxApp-metamodel elements are transformed to Screens-metamodel elements, so that detail is added. Each instance of a CRUD pattern that was introduced earlier is transformed to a `ListScreen`, `DetailsScreen` and `Entity`.

$T_{SigmaxApp2Screens}$ does not take into account the fact that the application under development is targeted for both smartphones and tablets. It applies simple transformation rules to derive the required screens from application patterns. The philosophy is that a `ListScreen` will always be a list screen, no matter what platform or target device the screen is meant for. A `ListScreen` simply has different implementations for smartphones and tablets.

A total of 11 transformation rules were defined, of which 2 were lazy rules. Lazy rules are only executed when they are called by another rule. Listing 5.1 shows the transformation rule `crud2screens`. This transformation rule transforms a CRUD-pattern to an `Entity`, `ListScreen` and `DetailsScreen` in a Screens-model. The properties set in an instance of a CRUD modeling element find their way into several modeling elements in the *Screens* model, or are used to derive properties.

```
1  rule crud2screens {
2   from crud:   SigmaxApp!CRUD
3   to  entity:   Screens!Entity (
4          name   <- crud.entityName,
5          fields   <- crud.properties->collect(p | thisModule.resolveTemp(p, '
       property')),
6          isHierarchical <- crud.isHierarchical
7         ),
8     detailsScreen: Screens!DetailsScreen (
9          id   <- entity.name + 'Details',
10         title  <- entity.name + 'Details',
11         entity  <- entity,
12         refScreens <- crud.references->collect(r | thisModule.doDetailsScreen(r))
13        ),
14    screen:   Screens!ListScreen (
15         id    <- entity.name + 'List',
16         title   <- crud.name,
17         entity   <- entity,
18         deletable  <- crud.deletable,
19         detailsScreen <- detailsScreen
20        )
21 }
```

Listing 5.1: Transformation rule `crud2screens`. This transformation rule breaks a CRUD-pattern up in Screens

Screens to Android. The $T_{Screens2Android}$ transformation is the second step in the transformation process. From the Screens-model that was created by the first model transformation, it derives a third model that is based on the Android-metamodel. We call this the Android-model, or $M_{Android}$ as it is referred to in Figure 5.5. While the previous model transformation was relatively simple, the $T_{Screens2Android}$ transformation adds many details and causes an explosion of classes. Figure 5.3 and 5.4 already suggested this, since the Android-metamodel recognizes significantly more elements than the Screens-metamodel.

Recall the Screens and Android metamodel from Figures 5.3 and 5.4, respectively. The relatively simple Screens-model must be transformed to a model that adds details so that the application under development can be properly represented for an Android platform. Table 5.2 shows how the Screens-metamodel elements are transformed to the proper Android-metamodel elements, and that indeed more detail is added.

Source element	Target element
Application	Application
Dashboard	LinearLayout Layout DashboardActivity
ListScreen	ListActivity Layout (2x) ListFragment FrameLayout FragmentContainer LinearLayout RelativeLayout (3x) FragmentContainer (3x)
...	...

TABLE 5.2: Partial transformation rules of how $T_{Screens2Android}$ transforms source elements from the Screens-metamodel to target elements for the Android-metamodel. We can clearly see that this transformation involves an explosion of classes, as a lot of detail is added.

Unlike the previous model-to-model transformation, $T_{Screens2Android}$ takes into account that the application under development is targeted for both smartphones and tablets. For the Android platform this means that multiple layout definitions are generated for different types of screens, each layout representing the same screen on different platforms.

A total of 24 transformation rules were defined, of which 12 were lazy rules. Listing 5.2 shows how the `ListScreen` that was created by the previous transformation is transformed to the proper Android modeling elements, including multiple layouts. This transformation clearly shows the complexity of $T_{Screens2Android}$. Two `Layout` definitions are added, one of which has the `target` property set to *default* and the other to *large*. On Android, tablets, for example, are classified as a 'large' layout. For both layouts, the required layout components are generated. For the large layout, the number of modeling components that are added in this transformation rule is substantially larger than for the default layout. Additionally, Listing 5.2 shows that from a single `ListScreen`, many modeling elements from the Android metmaodel are added, most of which are related to user interface components.

```
1  rule listscreen2activity extends screen2activity {
2   from scr:  Screens!ListScreen
3   to  act:  Android!ListActivity (
4         layoutDefinitions <- Sequence{ldD, ldL},
5         fragments   <- Sequence{
6               thisModule.details2android(scr.detailsScreen, false),
7               scr.detailsScreen.refScreens->collect(s | thisModule.details2android(s
      , true)),
8               fragList
9               },
10        menu    <- menu,
11        entity    <- scr.entity
12       ),
13    ldD:  Android!Layout (
14        target <- 'default',
15        content <- cDgroup
16       ),
17    ldL:  Android!Layout (
18        target <- 'large',
19        content <- cLgroup
20       ),
```

```
21      fragList: Android!ListFragment (
22          name    <- scr.entity.name + 'ListFragment',
23          contextMenuItems <- Set{if scr.deletable then thisModule.
         listFragmentAddDeleteSupport(act) else OclUndefined endif}
24        ),
25      menu:  Android!Menu (
26          name <- act.name + 'Menu',
27          items <- Sequence{itmCreate}
28        ),
29      itmCreate: Android!MenuItemOpenCreateEntityFragment (
30          id   <- act.name + 'Menu_Create_' + scr.entity.name,
31          title  <- 'Create ' + scr.entity.name,
32          fragment <- act.fragments->first(),
33          entity  <- scr.entity
34        ),
35      cDgroup: Android!FrameLayout (
36          views   <- Sequence{cDfragc}
37        ),
38      cDfragc: Android!FragmentContainer (
39          id    <- act.name + '_fragment_left', -- Same name as the first fragment in
         the large layout. Makes life easy. Trust me.
40          fragment  <- fragList
41        ),
42
43      cLgroup: Android!LinearLayout (
44          horizontal  <- true,
45          layout_width <- 'match_parent',
46          layout_height <- 'match_parent',
47          views   <- Sequence{cLrl1, cLrl2, cLrl3}
48        ),
49      cLrl1:  Android!RelativeLayout ( layout_weight <- 0.33, views <- Sequence{
         cLfrag1c} ),
50      cLrl2:  Android!RelativeLayout ( layout_weight <- 0.34, views <- Sequence{
         cLfrag2c} ),
51      cLrl3:  Android!RelativeLayout ( layout_weight <- 0.33, views <- Sequence{
         cLfrag3c} ),
52      cLfrag1c: Android!FragmentContainer ( id <- act.name + '_fragment_left',
         fragment <- fragList, next <- cLfrag2c ),
53      cLfrag2c: Android!FragmentContainer ( id <- act.name + '_fragment_center',
         fragment <- fragList, next <- cLfrag3c ),
54      cLfrag3c: Android!FragmentContainer ( id <- act.name + '_fragment_right',
         fragment <- fragList )
55  }
```

Listing 5.2: Transformation rule `listscreen2activity`. This transformation transforms a `ListScreen` to all required Android modeling elements.

5.4.2 Model-to-text transformation

A model-to-text transformation was created to transform an Android-model to the required source code. Figure 5.6 illustrates the operational context of the model-to-text transformation that was introduced. The green elements are provided by tools; the yellow elements were meta-models and transformations written during this work; the blue model is made by developers using the MDE environment; and the red elements are derived through transformations.

Android applications are packaged in *.apk*-files, which are Android package files. The most important contents of a package file are the compiled Java classes, XML definitions for screen layouts, a manifest file with information about the application, as well as graphical resources, such as icons. A package file can contain multiple layout definitions for a single screen, from

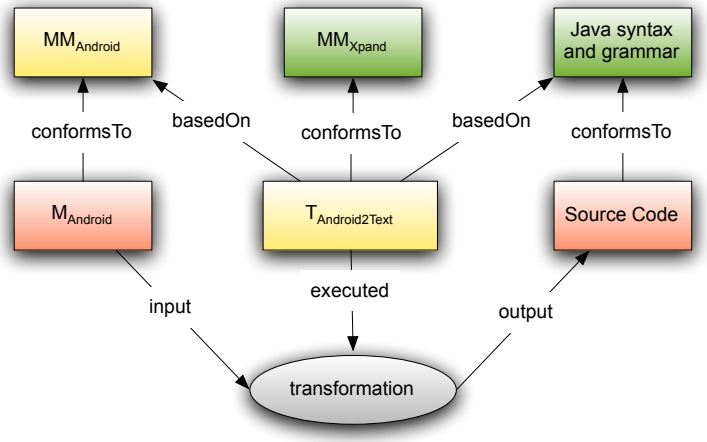

FIGURE 5.6: Operational context of the model-to-text transformation.

which Android automatically selects the correct layout definition based on the screen size at runtime. This allows developers to deploy a single package file to multiple targets.

The model-to-text transformations are where most user interface design guidelines and principles were implemented. Most of the guidelines have to do with application behavior, which is eventually determined by code. Since guidelines should always be taken into account, adding them as modeling elements would be superfluous and increase the complexity of modeling and transformations. The guidelines that were incorporated were: "*be consistent*", "*provide understandable feedback*" and "*be supportive and minimize manual input*". *Be consistent* is inherent to adopting a Model-Driven Engineering environment. The layouts that are generated for each pattern follow the same transformation rules, thus making the layout consistent. *Providing understandable feedback* was realized in multiple places. When an entity is created, updated or deleted, a short Android-compliant notification is shown, informing the user of what happened. *Being supportive and minimize manual input* was realized using a specific type of keyboards that is made available on Android to only allow numbers as input in text fields. The Android model that is used to generate the source code is aware of which properties are associated with an entity. Thus, for numerical properties, a *numerical* property is provided to the Android component to which it is transformed. Listing 5.3 shows this model-to-text transformation.

```
1 «DEFINE view FOR android::layout::view::EditText»
2  <EditText
3   «IF inputType.length > 0»android:inputType='«inputType»'«ENDIF»
4   «EXPAND arguments FOR this»
5   />
6 «ENDDEFINE»
```

Listing 5.3: Model-to-text transformation that shows a numerical keyboard when the `EditText` element is activated, provided that `inputType` is set to `numeric`.

A total of 85 model-to-text `DEFINE` blocks were written, divided over 5 different template files. Listing 5.4 illustrates the complexity that is associated with the model-to-text transformation. The listing begins with the `DEFINE` block `main`, executed in the context of an `Application` modeling component. This is the first model-to-text transformation definition that is executed.

It creates the manifest file, and makes sure the `activity` block is executed for all activities. In the `activity` block, we see that two Java-files are created: `Base«name».java` and `«name».java`. All generated code is added in `Base«name».java`, and an empty `«name».java` is created. The base class is abstract, and the extending class is always constructed. This allows developers to extend or modify the generate code. At the end of the `activity` block, all layout definitions belonging to the activity are generated. This is where the support for multiple target platforms is realized.

```
1  «DEFINE main FOR Application»
2   «setAppIdentifier(identifier)»
3   «setPackageName(identifier)»
4   «FILE 'AndroidManifest.xml'»
5    ...
6   «ENDFILE»
7   «EXPAND activity FOREACH activities»
8  «ENDDEFINE»
9
10 «DEFINE activity FOR activity::Activity»
11  # We put everything in the base activity, so that it can be overriden/extended by
       the user if desired.
12  «FILE 'src/' + getPackageDir() + '/activities/Base' + name + '.java'»
13  package «getPackageName()».activities;
14
15  «classHeader('Base' + name + 'Activity')»
16  abstract class Base«name» extends Activity«EXPAND activityImplements FOR this» {
17   // Whether this activity uses multipane or not
18   protected boolean isLarge = false;
19
20   @Override
21   public void onCreate(Bundle savedInstance) {
22    ...
23    isLarge = ((getResources().getConfiguration().screenLayout & Configuration.
       SCREENLAYOUT_SIZE_MASK) >= Configuration.SCREENLAYOUT_SIZE_LARGE);
24    ...
25    «EXPAND activityOnCreateBehavior FOR this»
26   }
27
28   «EXPAND menu::methods FOR this»
29   «EXPAND activityMethods FOR this»
30   «EXPAND activityBehavior FOREACH behaviors»
31  }
32  «ENDFILE»
33
34  # This activity can be modified by the user. Should have some check that it is not
       overwritten tho ....
35  «FILE 'src/' + getPackageDir() + '/activities/' + name + '.java'»
36   package «getPackageName()».activities;
37
38   «classHeader(name + 'Activity')»
39   public class «name» extends Base«name» {
40    public «name»() {
41     super();
42    }
43   }
44  «ENDFILE»
45
46  # Create activity layout files. We can have more than one to support multiple
       devices.
47  «FOREACH layoutDefinitions AS layout»
48   «FILE 'res/layout' + (layout.target == 'default' ? '' : '-' + layout.target) + '/
       ' + name.toLowerCase() + '.xml'»
49    <?xml version='1.0' encoding='utf-8'?>
50    «EXPAND layout::content FOR layout.content»
51   «ENDFILE»
```

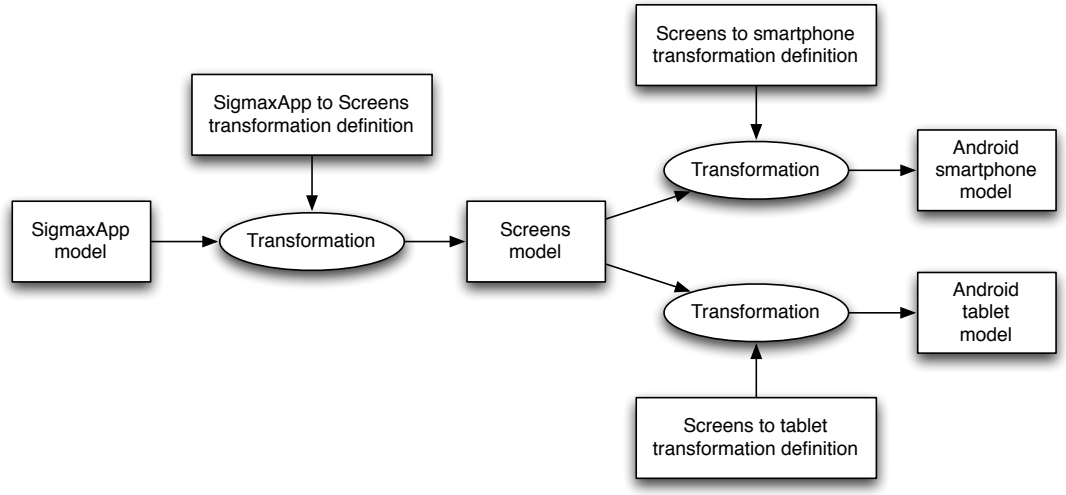

FIGURE 5.7: Alternative Android model generation from the SigmaxApp model.

```
52    «ENDFOREACH»
53
54    «EXPAND fragment FOREACH fragments»
55    «EXPAND menu::menu FOR menu»
56  «ENDDEFINE»
```

Listing 5.4: First model-to-text transformation rules that are executed.

5.5 Discussion

Our MDE solution contains three metamodels, thus at least two model-to-model transformations were required. In addition, a model-to-text transformation is required to generate source code. The aim of the environment is to be able to generate mobile applications for multiple target platforms. Alternatively to the method described in this chapter, a second model-to-model transformation could be developed that derives two Android models from a Screens model: one aimed for smartphones and one aimed for tablets, as illustrated in Figure 5.7. The advantage is that two models are generated, which are focused at a specific target, allowing to model platform specifics. The disadvantage is that we would have two models and two model transformation definitions to maintain. Additionally, the Android platform provides sophisticated support for the development of multi-target applications using a single code base, hence we only need a single model. To generate an application that is as much as possible in line with the developer principles of the target platform, a single Android model was generated.

Another point of discussion may be a bit odd, but why would we need model transformations at all? From a single source model it is possible to immediately generate the required source code for each desired platform using a single model-to-text transformation. In this case we would not need to develop two more metamodels or write additional model transformation, but would only have one complex model-to-text transformation. However, the complexity would be very difficult to maintain. Splitting up the model in multiple transformations keeps complexity

manageable, thus eliminating these transformations would introduce a lot of complexity. For that reason, we developed an environment as described in this chapter.

The current approach allows to easily switch to different platforms. If the same application must be transformed to an iPhone implementation, two actions need to be performed. First, a metamodel is required that can describe the application for an iPhone environment. Second, model transformations need to be written to transform a platform-independent Screens-model to a platform-specific iPhone model.

The project workspace that contains all metamodels, model-to-model and model-to-text transformations, as well as a sample source model can be downloaded from `https://github.com/markoudev/modeling/tree/thesis`.

Chapter 6

Acceptance and evaluation

This chapter assesses the acceptance of the developed proof of concept implementation of a Model-Driven Engineering approach for mobile devices, which takes into account user interface design guidelines. By means of a case study and semi-structured expert interviews with experienced application architects, we show that a model-driven approach may help to significantly reduce development time.

6.1 Case Study

To elaborate on the details of the models and model transformations, and to show that the implemented solution design from Chapter 5 works, a case study was conducted. The requirement of the case study was to implement an Android application that features as much functionality available in existing Sigmax applications.

Sigmax develops the *Field Mobility Suite* (FMS), an application that is used by field workers for planning and registration of articles they used during tasks. The FMS is an abstract application, so that it can be tailored to meet requirements for as many customers as possible. Many customers have comparable requirements for which the same software components may be used in order to reduce development time. For this case study, five primary types of entities from the FMS application were considered: *tasks*, *articles*, *problems*, *objects* and *locations*. We discuss each of these entities in more detail:

1. **Tasks** provide information about the work a field worker has to perform. They can be assigned to a field worker by a planner and automatically sent to the mobile device of the field worker.
2. **Articles** can for example be parts or tools that are used during the execution of a task. When a task was conducted by a field worker, he or she registers the articles that were

used during the task. This information is then synchronized, and may be used to keep a proper stock, as well as for billing and administration purposes.

3. **Problems** need to be solved and are assigned to a task so that they are properly registered. The registration of problems may help field workers and customers to resolve comparable problems faster and more efficiently.
4. **Objects** are physical objects in the field that are maintained by field workers. Objects may be hierarchical, allowing to break down an object into detailed parts in order to be as precise as possible.
5. **Locations** indicate where a certain task or object is located so that field workers know where they have to go to. Locations are hierarchical, so that countries, provinces, cities and so forth can be properly structured.

The case study is limited to creating, reading, updating and deleting instances of each of these entities on the mobile device itself. Code generation for synchronization protocols was not in the scope of this research.

In the remainder of this section we discuss the SigmaxApp-model that was created for this case study. Furthermore the results of both model-to-model transformations and the model-to-text transformation are discussed. To increase readability, this chapter focuses on how the *Tasks* entity is modeled and transformed throughout the transformations. Discussing each of the entities would be repetitive and reduce readability.

6.1.1 Application model

Figure 6.1 shows the source model for this case study. The bold printed text represents the metamodel element that was instantiated, and the most important properties for each of these elements are printed in italic. This figure focuses on the *Tasks* implementation of a CRUD-pattern and shows which descriptives are associated with *Tasks*. The other four CRUD-patterns have similar properties, but are left out for illustration purposes.

An application with the name *FMS Port* was modeled using the *SigmaxApp*-metamodel. The application recognizes five CRUD-patterns so that *tasks*, *articles*, *problems*, *locations* and *objects* can be created, read, updated and deleted. Tasks have six different properties: a *title*, a *description*, an indicator whether the task is *in progress*, a list of *used articles* and a reference to the *solved problem*. Articles that were used during the task have two extra properties: an *amount* of how many items of some article were used, and a *reason* for its use.

6.1.2 Transformations

The source model is based on the SigmaxApp-metamodel. As presented in the solution design from Chapter 5, this source model is transformed to a Screens model, which is again transformed to an Android model. From this Android model, source code is generated. Each of these transformations is discussed in more detail in this section, where the focus is on how the *Tasks* CRUD-pattern is transformed throughout the process.

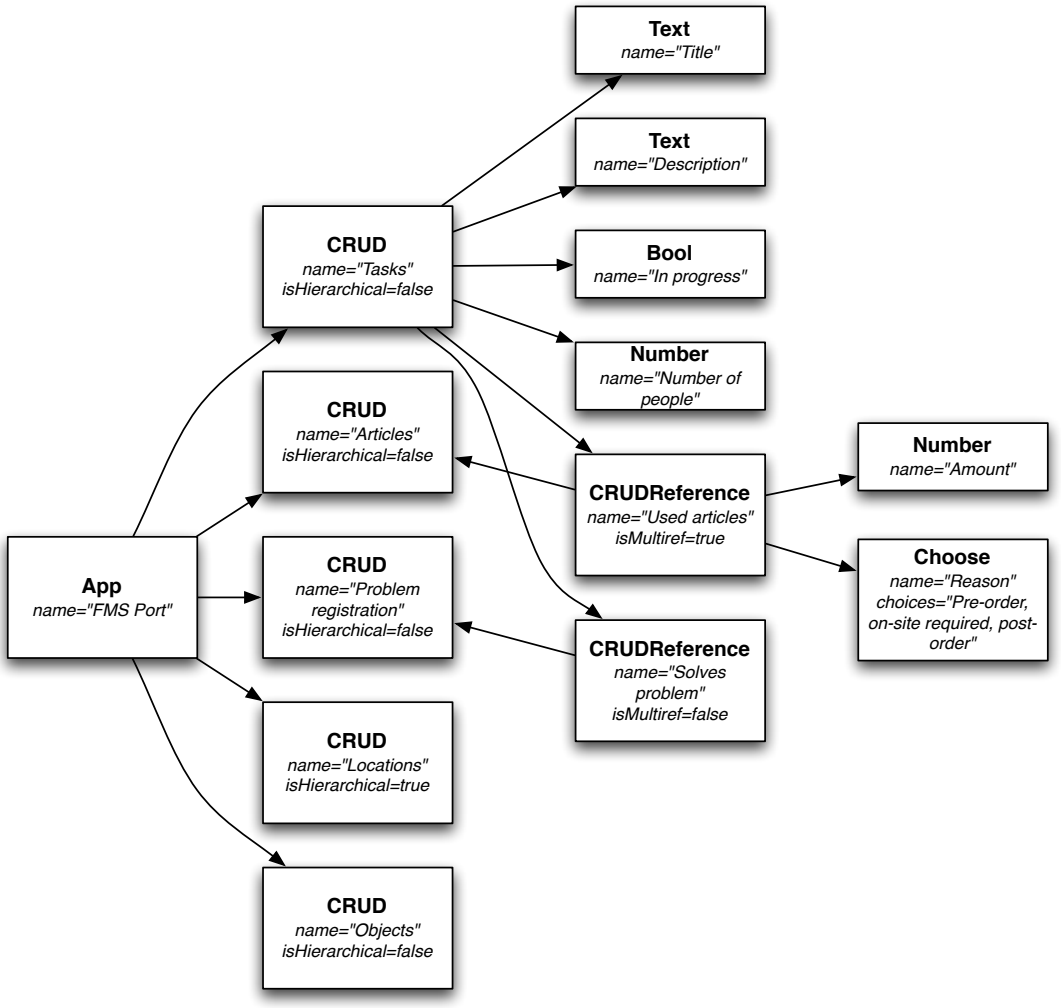

FIGURE 6.1: The SigmaxApp-model for the case study.

SigmaxApp to Screens. As described in the solution design in Chapter 5, a model based on the SigmaxApp metamodel is transformed to a model based on the Screens metamodel. A Screens-model represents the application in terms of types of screens and their relations. Figure 6.2 shows the most important parts of the Screens model that was derived from the SigmaxApp model as it was described earlier in Section 6.1.1. Figure 6.2 focuses on the model elements that are derived from the *Tasks* CRUD-pattern.

From the SigmaxApp model, the `App` model element was transformed to an `Application` model element in the Screens model. For the *Tasks* CRUD-pattern that was present in the SigmaxApp-model, a `ListScreen` and a `DetailsScreen` were directly created in the Screens model. The `ListScreen` allows viewing a list of tasks, and the `DetailsScreen` describes a screen that can be used to create new tasks, and updating or viewing existing ones. Deleting tasks may be facilitated by the `ListScreen`, but this functionality is not specified in terms of screens. Additionally, an `Entity` was created in the Screens model to represent a *Task*. This model element was created from the CRUD-pattern in the SigmaxApp model. Figure 6.2 shows

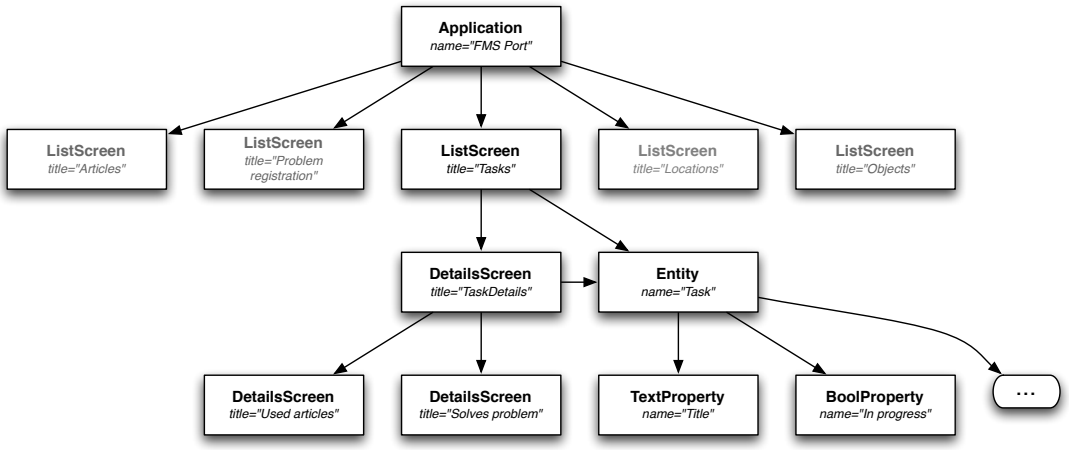

FIGURE 6.2: The generated Screens model for the case study. Several `Property` elements have been left out for readability, as well as how other `ListScreens` were structured.

that the entity has a `TextProperty` and `BoolProperty`, among other properties. More properties were created, but left out of the figure for readability.

Because the *Tasks* CRUD-pattern in the SigmaxApp model for the case study uses two `CRUDReferences`, two more `DetailsScreens` were created and linked to by the *TaskDetails* details screen. These two details screens allow a *Task* to be linked to *Articles* or a *Problem*, while creating or updating a task.

Screens to Android. The second model transformation involves transforming the Screens model to an Android model. This transformation adds the required elements to an Android model, in order to represent the application as it would be implemented for the Android platform. Figure 6.3 shows the most important part of the Android model, and clearly shows the complexity of modeling an application for the Android platform. Figure 6.3 focuses on the model elements that were added from the original *Tasks* CRUD-pattern. It clearly shows that different layouts were generated for a single Android activity. *EClasses* denoted with an 'A' represent Android elements. Other items were introduced to add required detail or implicitly describe Android concepts.

As Figure 6.3 already suggests, this transformation causes an explosion of modeling elements and derives a complex model from the relative simple Screens model. Since the Screens model does not take multiple target mobile devices into account, this model transformation does. We can see that from the Screens-model, the *ListScreen* for tasks is transformed to a *ListActivity* in the Android model. This activity is a specialization of an `Activity`, so that during the model-to-text transformations we know what code to generate for this kind of activity. The `ListActivity` "*TaskListActivity*" forms the root of what in the SigmaxApp model was added as a CRUD-pattern.

The `ListActivity` recognizes two *Layout* definitions, for which the target property was set to *default*, and the other to *large*. The default layout describes a layout that is meant for smartphones, while the large layout describes a layout that is meant for tablets.

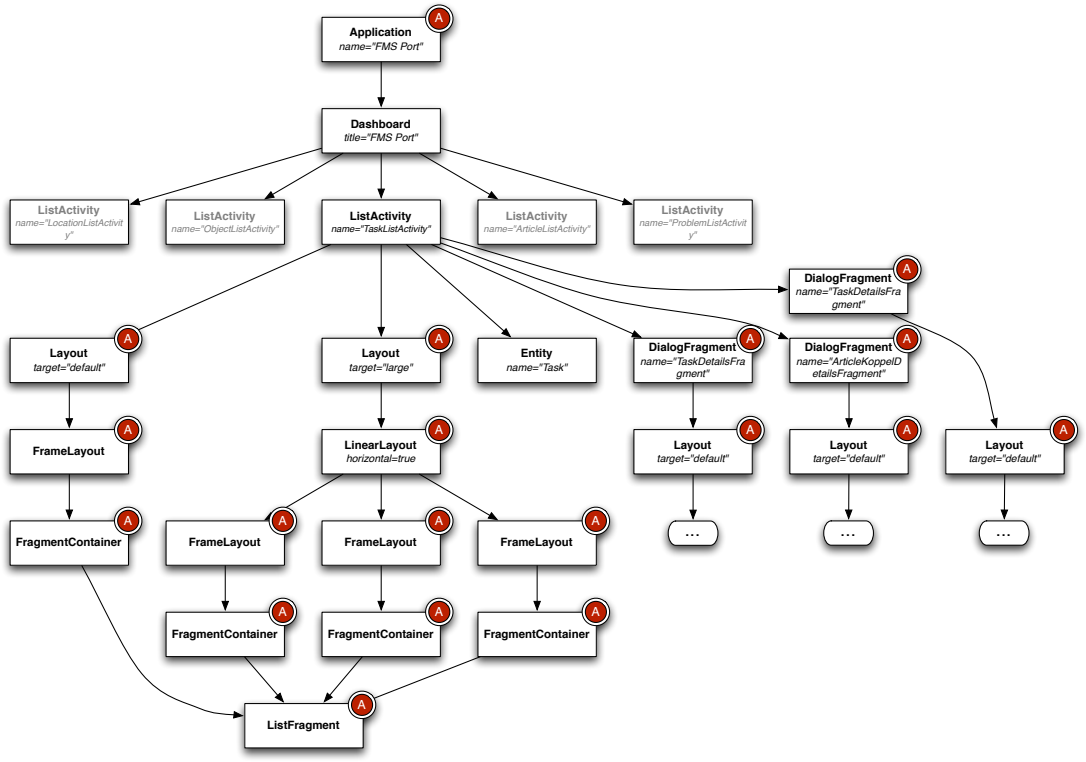

FIGURE 6.3: The generated Android model for the case study. Several elements have been left out for readability.

Android model to source code. From the Android model we can generate source code using a model-to-text transformation. The model-to-text transformation generates the required Java classes, as well as XML-files that are required to describe different layout definitions, menu structures or collection of strings. A total of 68 files were generated. 38 Java classes were generated, 12 for activities, 12 for fragments, and 14 for data storage support. Additionally, 30 XML-files were generated: 23 for layout definitions, 5 for menu structures, one that contains string collections, and one Android manifest file.

All Java code was generated as an abstract class. For each of the generated abstract classes, an empty class was generated that *extends* the base class. For example, the Java code that was generated for the `ListActivity` is the abstract base class `BaseTaskListActivity`. The empty class `TaskListActivity` then subclasses the abstract base class. This pattern allows the generated code to be modified and extended by programmers if desired.

6.1.3 Result

The generated source code was compiled and packaged for the Android platform, and deployed on an Android smartphone. Figure 6.4 shows the smartphone implementation of the source model for the case study.

Figure 6.4(a) shows the *Dashboard* that allows navigation to the five CRUD-patterns. Figure 6.4(b) shows the Android implementation of a *ListScreen* including some *Tasks*. Clicking an item in this list opens up a *DetailsScreen*, as can be seen in Figure 6.4(c). Figure 6.4(d) shows a *DetailsScreen*

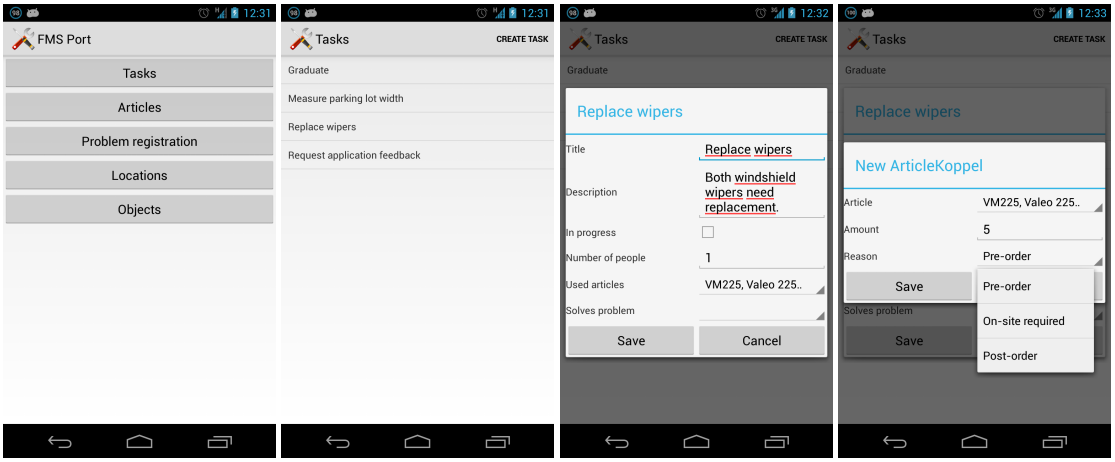

FIGURE 6.4: Screenshots of the Android smartphone implementation of the case study.

to add a used article to a task. This screen allows to set the additional properties that we added in the source model to be set.

6.2 Expert interviews

As this research adopted a pragmatic approach, expert interviews at Sigmax were conducted in order to assess the acceptance the work, and to determine whether the solution design is useful in the development of applications for mobile devices.

This section provides an overview of the most important aspects of the setup and discusses the results. Appendix D contains a complete overview of the interview questions and results.

6.2.1 Setup

The expert interviews were designed to be semi-structured, and took about 30-45 minutes. The interview participants were experts on the subject of software development and software architecture within Sigmax. Four people were selected as participants, based on their skills and role within Sigmax. Before the interviews were conducted, a presentation of what MDE exactly is and a demonstration of the implemented MDE environment was given. Questions asked were, amongst others, "*for what goal do you think MDE is best applicable?*", "*what do you think of the source model, as it was presented?*", "*what do you think of the separation of models into three levels?*" and "*do you think the proposed MDE environment would help to reduce development time for Sigmax applications?*". During the interview, audio recordings and textual notes were made. The results of the interviews were written down according with the *selective reading approach*, after which open coding in grounded theory was applied to analyze the results. This approach is similar to the expert interviews that were used as a data gathering technique for the design guidelines, as presented in Section 4.1.2. The data units that were used in the grounded theory approach were textual notes and expressions by the participants.

6.2.2 Results

This section discusses the most important results of the expert interviews. A complete overview of the results can be found in Appendix D. Table 6.1 shows the data units that were answered the same by at least two participants.

In general, experts were positive about the concept of Model-Driven Engineering and believe that MDE can be used for the development of proof of concepts or mockups. They clearly see the benefits of an MDE approach, such as standardization and consistency of application development, as well as faster development and focus on the design instead of implementation. Provided that an MDE environment is adopted and provides the proper modeling elements and transformation, experts agree that development using MDE is significantly faster.

While many positive comments were provided during the interviews, experts independently agree that a Model-Driven Engineering approach introduces certain drawbacks to the organization. The most occurring comment is that an MDE development may be difficult to adopt in an existing development process, and to get it accepted by developers. This is caused by several issues such as the ability of the MDE environment to be able to integrate with other development tools, but also the ability of developers to understand the level of abstraction introduced by MDE. Additionally they fear that if the metamodel that provides modeling elements for the source model is not complete enough, developers will miss freedom. Experts also mentioned that the generated source code should be easy to extend.

Experts found the proposed environment to be potentially valuable, but argued that in its current state it is not complete enough to be adopted. However, all participants agreed that the source model brings structure to applications by providing application patterns. Two participants agreed that the proposed source model would help in the development of multi-target mobile applications. A drawback of the current source model is that it currently does not support the addition of specific bits of logic. Most experts agreed that breaking up the model transformations in multiple transformations is arbitrary, but indeed helps manage transformation complexity, something that is seen as a positive.

6.3 Discussion

The complexity associated with modeling an application for the Android platform including details required for code generation can be seen in Figure!6.3. By defining model transformations and multiple metamodels, this complexity can be captured in a much simpler model, as our SigmaxApp-model. We saw that experts agreed on the benefits of such an approach, such as more consistent and faster development, but also more focus on the design instead of implementation.

During the expert interviews, some experts argued that a good alternative to the proposed model chain would be to derive a Screen-model for each target platform. Figure 6.5 illustrates this alternative design. This figure illustrates that a separate screens model could be derived for each target platform, providing more flexibility in user interface modeling. Experts argued that it is difficult to talk about *screens* when you do not know your target platform. It may be

+/-/!	Sentence, claim (data unit)	1	2	3	4	Total
+	MDE is useful for quick proof of concepts or mockups	x	x	x	x	4
+	The proposed source model helps to structure applications	x	x	x	x	4
+	The proposed environment may help Sigmax if it is extended	x	x	x	x	4
!	The generated code should allow customization and extensibility	x	x	x	x	4
+	MDE brings consistency in development and makes it predictable	x	x		x	3
+	MDE prevents repetitive work	x	x	x		3
-	MDE may be difficult to adopt and get accepted in an existing development process	x	x		x	3
-	If the source model isn't good enough, developers miss freedom		x	x	x	3
+	MDE is useful for the development of multi-platform applications	x		x	x	3
+	Transformation separation helps to manage complexity	x	x	x		3
+	MDE allows for faster development.		x		x	2
+	MDE helps to focus on the design, you are not distracted by the details		x	x		2
+	MDE brings standardization			x	x	2
-	MDE uses a level of abstraction not easily understood by all	x			x	2
-	To setup an MDE environment may be difficult		x		x	2
-	Debugging the generated application becomes difficult		x	x		2
-	The adaptability or extensibility of generated code may be difficult	x	x			2
-	It is difficult to find the right level of abstraction for a source model		x	x		2
+	Proposed source model helps to develop multi-platform	x	x			2
-	Talking about screens means you have to know the target platform	x		x		2
!	Instead of generating for Android, you could generate for a framework		x		x	2

TABLE 6.1: Answers and claims by expert interview participants that were given by at least two participants. The characters +/-/! indicate whether it is a positive aspect, negative aspect or point of attention for an MDE environment. The headings *1-4* shows which participant made the corresponding claim. The *total*-header shows the number of participants that made the claim. The Table is sorted descending on the number of people that made a claim.

feasible for smartphones and tablets, but what if the same source model is to be transformed to web pages, experts argued. It is true that deriving a Screens-model meant for multiple target platforms limits the use of platform-specific screens, and limits the number of target platforms for which the proposed environment can generate code for. Generating a separate Screens-model for each target platform also provides more flexibility to describe user interfaces for each platform. A drawback of deriving a Screens-model for each target platform is that a separate application is eventually generated for each target platform. While this research focused on the generation of mobile applications for smartphones and tablets, a single Screens-model can be used. Because Android was chosen as the target platform, this was also more in line with how Android applications are developed. However, to be prepared for future developments, an alternative approach such as the one shown in Figure 6.5 may provide more flexibility.

If the proposed environment was to be extended with multiple application patterns, and the

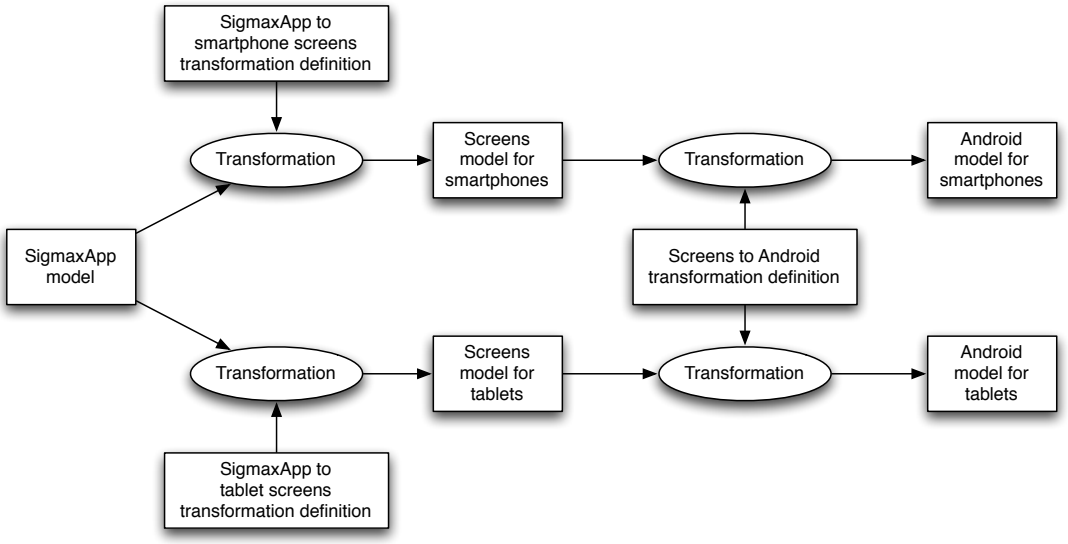

FIGURE 6.5: An alternative transformation chain for a Model-Driven Engineering environment, argued by experts during expert interviews.

generated application was visually more appealing, experts say that modeling applications may help Sigmax in the development of mobile applications. The goal of the mobile applications generated by the MDE environment, however, is seen more as to be able to quickly generate a proof of concept or application mockup. If an MDE environment was to be adopted for the development of mobile applications, it may be better to develop or adopt an application framework in which only the details of the application have to be filled.

Chapter 7

Conclusions and final remarks

This chapter concludes the research, and provides an answer to the presented research questions. Suggestions for future work are also provided.

7.1 Conclusions

This section presents the conclusions of our work by answering the research questions that were introduced in Chapter 1, starting with question *SQ1*:

> **SQ1.** *How can Model-Driven Engineering be applied during the development of mobile applications, in order to speed up development and cope with a variety of mobile devices?*

Chapter 2 presented the results of a literature study on the topic of Model-Driven Engineering and how MDE can be applied in the development of multi-target mobile applications. We learned that the core principles and viewpoints of MDE are very suitable to support multi-target application development. By using a single source model, defining the proper metamodels for each desired target platform, and developing proper model-to-model and model-to-text transformations, MDE can in theory be tailored to meet the requirements of any target platform. In practice this means that the correct metamodels and transformations must be created, in order to support the development process.

> **SQ2.** *Which are the challenges in user interface design and what are the usability issues for mobile applications?*

Chapter 3 presented the common challenges in user interface design for mobile applications. Next to hardware challenges that primarily concern input and output limitations, software challenges are related to the smaller screen, navigation through menus and data input. The smaller screen forces developers to make a trade-off between what should and should not be displayed.

Navigation through menus is especially difficult in hierarchical menus. Another software challenge is inputting data: as data input is more difficult, specific input paradigms could be made available in software. Finally, poor network coverage introduces issues if a mobile application requires an Internet connection. In case of errors, the error messages should be understandable for users.

Chapter 4 focused more on challenges inspired by Sigmax applications. From expert interviews and field studies we learned that the primary issues are related to the quality of the devices as well as their responsiveness. Users often expected a higher quality from the device in terms of physical appearance and the image quality of pictures made using the device camera. During its use, it was found that people had to scroll through many long lists of items. A challenge here is to tailor the list so that the number of items displayed can be decreased.

> **SQ3.** *Which user interface design guidelines should be taken into account during the development of mobile applications for different target platforms?*

Chapter 3 introduced UI guidelines and principles from literature. An important aspect to take into account during the design process, are success factors. *Usability*, *fit for mobile work context* and *positive impact on work productivity* are key factors for the success of mobile applications in professional environments [56, 58]. These factors guided the development of design guidelines.

Chapter 4 presented the adopted and developed guidelines. Seven guidelines were developed. The guidelines that were incorporated, were to *be consistent*, *provide understandable feedback* and *be supportive and minimize manual input*. To *be consistent* is meant to delivering a consistent user experience while using different screens of an application, or using the application on different platforms. This concerns several dimensions, such as the types of screens that are used, use of language, but also the structuring of information. During the field studies, we observed that some people were confused by comparable types of screens that did not behave as expected. *Provide understandable feedback* was incorporated because during the initial expert interviews and later field studies it was found that when the device was unable to complete it task, technical error messages were displayed. These messages are not very well understood by most users, causing them to think that they did something wrong. The guideline was developed to instruct developers to provide understandable error messages that clearly inform users of what happened and what they can do about it. *Be supportive and minimize manual input* reduces the work load for users by assisting the user as much as possible. For example, if text input is required but only numbers are required in the field, a numerical input mechanism should be used. If the current address is required and the device offers GPS capabilities, assist the user by proposing the current location as the most likely address.

> **RQ.** *How can a Model-Driven Engineering environment be developed to increase the consistency, usability and development speed of mobile applications, while taking into account user interface design guidelines?*

In Chapter 5 we presented the solution design for an MDE environment for the development of multi-target applications that takes into account a selection of the user interface design guidelines from Chapter 4. By introducing a metamodel that allows developers to create a single

source model for an application for mobile devices, and metamodels to describe an Android application for multiple platforms, we showed that MDE indeed can be used for the development of multi-target mobile applications on the Android platform. Alternatively a metamodel for a different target platform could be developed, and new transformation rules could be written accordingly in order to adopt a new platform. The introduction of the Screens model and therefore multiple model-to-model transformations, helps maintain the complexity of the model transformations from the source model to a model that represents the application under development for a specific target platform into multiple levels.

The user interface design guidelines that were introduced in Section 4.2 primarily consisted of high-level guidelines and principles, of which some were adopted by the MDE environment. Considering the guideline *be consistent*, this is inherent to a Model-Driven approach, since source code is generated using transformation rules that are executed more than once. Considering the guideline *provide understandable feedback*, each instance of a CRUD-pattern in the source model an interface to a data storage is generated. Whenever an entity is created, updated or deleted, a short notification is displayed, informing the user of the action that was performed. Considering the guideline *be supportive and minimize manual input*, numerical properties added to an instance of a CRUD-pattern in the source model are ultimately transformed to a text box on Android that when activated shows an on-screen keyboard that only allows numerical input.

However, there are also some disadvantages and challenges to be addressed before an MDE environment can be adopted. The first challenge concerns a business aspect. Companies have to deal with budgets, and developing an MDE environment suited to support the development department of a company may be a time consuming and costly investment, as argued by experts during validation interviews. Additionally, developing an MDE environment that is integrated with other development environments and tools may be a challenging task.

Technically speaking, it remains difficult to find the most suitable level of abstraction for a source model. There is not a single correct answer to the question which level to choose. The level has to be chosen by carefully weighing the pros and cons of each abstraction level. Finally, experts argue that not all developers have the ability to adopt the level of abstract thinking required by MDE.

7.2 Future work

While the proof of concept shows that a development approach using MDE that takes into account user interface design guidelines is feasible, it is far from complete. To adopt the proposed environment, the source metamodel should be extended with modeling elements that allow to model an application that could be used in the field. For future research we propose the following topics that may be interesting to investigate further.

Graphical visualization of the source model. Something that was not included in this research, but may be of interest for future work, is the visualization of the source model. A Domain-Specific Language (DSL) allows both developers and designers to invest in domain

knowledge instead of technical (programming) languages. The EMFText project [8] facilitates the definition of textual DSLs for languages described as an Ecore metamodel. They can be represented using a textual concrete syntax, or a graphical concrete syntax. However, this graphical syntax is still a diagram that looks like a UML class diagram. Especially when working with user interfaces, such as in this work, a graphical representation of the source model is desired. We therefore argue that one could investigate how a graphical concrete syntax for a source model can be adopted, in order to graphically model user interfaces at an abstract level, and what this graphical syntax would look like.

Abstraction levels of application development. In the solution design of this work we introduced three different metamodels that describe the application at different levels of abstraction, and from different viewpoints. The choice for these three levels was argued by some experts to be arbitrary. While this may be the case, the choice for three different metamodels was argued carefully in this work, provided the context of the proof of concept. It may be interesting to investigate which factors influence the decision of developing a certain amount of metamodels and transformations. If such factors can be identified, the choice for the number of metamodels and transformations can be better justified.

Balance between generating components and using a framework. The model-to-text transformation in this work generated many software components without the use of a framework. While any target platform can be seen as a framework, since platforms provide a sophisticated API, software developers often use an application framework to simplify application development. For future research it may be interesting to investigate whether it is more beneficial to invest in the development of a framework to be used by an MDE environment. It can be argued that adopting a framework simplifies the model-to-text transformations, since complex source code can be captured by an application framework. This way, functionality that could be facilitated by a framework does not have to be generated every time with model-to-text transformations.

Allow customization of generated code. During the expert interviews it was found that all interviewed experts argued that the generated code should allow customization and extensibility. While this is possible by generating abstract base classes and extending those, as it was done in this work, it may be interesting to investigate other possibilities. It is for example, interesting to investigate whether specific application source code may be captured in the source model to ultimately be used in code generation. The challenge is, however, to cope with different target platforms that use different programming languages. If such an approach were possible it would save development time, as specific source code can be used in code generation for multiple platforms, and only needs to be written once.

Bibliography

[1] Christine Anderson, Sandra G. Hirsh, and Andre Mohr. Wheels around the world: Windows Live mobile interface design. In *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, pages 2113–2128, New York, NY, USA, 2008. ACM.

[2] Apple. iOS Human Interface Guidelines, 2010.

[3] Nathalie Aquino, Jean Vanderdonckt, and Oscar Pastor. Transformation templates: adding flexibility to model-driven engineering of user interfaces. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1195–1202, New York, NY, USA, 2010. ACM.

[4] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *Software, IEEE*, 20(5):36 – 41, sept.-oct. 2003.

[5] Florence Balagtas-Fernandez, Jenny Forrai, and Heinrich Hussmann. Evaluation of user interface design and input methods for applications on mobile touch screen devices. In Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, Raquel Prates, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2009*, volume 5726 of *Lecture Notes in Computer Science*, pages 243–246. Springer Berlin / Heidelberg, 2009.

[6] Jean Bézivin. On the unification power of models. *Software and Systems Modeling*, 4:171–188, 2005.

[7] Luca Chittaro. Visualizing information on mobile devices. *Computer*, 39(3):40 – 45, march 2006.

[8] Christian Wende and Mirko Seifert and Florian Heidenreich and Sven Karol and Jendrik Johannes. EMFText website. Accessed on November 18, 2012.

[9] Juliet M Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1):3–21, 1990.

[10] Henry Been-Lirn Duh, Gerald C. B. Tan, and Vivian Hsueh-hua Chen. Usability evaluation for mobile device: a comparison of laboratory and field tests. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, MobileHCI '06, pages 181–186, New York, NY, USA, 2006. ACM.

[11] Mark Dunlop and Stephen Brewster. The Challenge of Mobile Devices for Human Computer Interaction. *Personal and Ubiquitous Computing*, 6:235–236, 2002.

[12] Eclipse Foundation. Eclipse. http://eclipse.org/.

[13] Eclipse Modeling. Ecore. http://wiki.eclipse.org/Ecore.

[14] Eclipse Modeling. Model To Text (M2T).

[15] Eclipse Modeling. Modeling Project.

[16] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods*, 35:379–383, 2003. 10.3758/BF03195514.

[17] Ben Fehnert and Alessia Kosagowsky. Measuring user experience: complementing qualitative and quantitative assessment. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, MobileHCI '08, pages 383–386, New York, NY, USA, 2008. ACM.

[18] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[19] Gaëlle Calvary and Joëlle Coutaz and David Thevenin and Quentin Limbourg and Laurent Bouillon and Jean Vanderdonckt. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

[20] Gartner. Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012, February 2013. Accessed on March 28, 2013.

[21] Arjan Geven, Reinhard Sefelin, and Manfred Tscheligi. Depth and breadth away from the desktop: the optimal information hierarchy for mobile use. In *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, MobileHCI '06, pages 157–164, New York, NY, USA, 2006. ACM.

[22] Jun Gong and Peter Tarasewich. Guidelines for handheld mobile device interface design. In *Proceedings of DSI 2004 Annual Meeting*, 2004.

[23] Google. Android Design, 2012.

[24] Google. User Interface Guidelines, 2012.

[25] Jack Greenfield and Keith Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, pages 16–27, New York, NY, USA, 2003. ACM.

[26] Object Management Group. MDA Guide Version 1.0.1, 2003.

[27] Jonna Häkkilä and Jani Mäntyjärvi. Developing design guidelines for context-aware mobile applications. In *Proceedings of the 3rd international conference on Mobile technology, applications & systems*, Mobility '06, New York, NY, USA, 2006. ACM.

[28] Kuo-Ying Huang. Challenges in Human-Computer Interaction Design for Mobile Devices. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 1, October 2009.

[29] Ivan Kurtev and Luís Ferreira Pires. Lecture sheets for Advanced Design of Software Engineering: Model-Driven Engineering, 2012.

[30] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of Computer Programming*, 72(1–2):31 – 39, 2008.

[31] Frédéric Jouault and Ivan Kurtev. On the architectural alignment of ATL and QVT. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1188–1195, New York, NY, USA, 2006. ACM.

[32] Stuart Kent. Model driven engineering. In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer Berlin / Heidelberg, 2002.

[33] Woo Yeol Kim, Hyun Seung Son, Jae Seung Kim, and Robert Young Chul Kim. Adapting model transformation approach for android smartphone application. In Tai-hoon Kim, Hojjat Adeli, Rosslin John Robles, and Maricel Balitanas, editors, *Advanced Communication and Networking*, volume 199 of *Communications in Computer and Information Science*, pages 421–429. Springer Berlin Heidelberg, 2011.

[34] Thomas Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5:369–385, 2006.

[35] Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. Model-based DSL frameworks. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 602–616, New York, NY, USA, 2006. ACM.

[36] Steve Love. *Understanding Mobile Human-Computer Interaction*. Information Systems Series. 2005.

[37] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past, Present, and Future of Model-Based User Interface Development. *i-com*, 10(3):2–11, November 2011.

[38] Microsoft. Design Guidelines for Windows Mobile 6.5, 2010.

[39] Microsoft. User Experience Design Guidelines for Windows Phone, 2012.

[40] Jakob Nielsen and JoAnn T Hackos. *Usability engineering*, volume 125184069. Academic press Boston, 1993.

[41] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, CHI '90, pages 249–256, New York, NY, USA, 1990. ACM.

[42] Erik G. Nilsson. Design patterns for user interface for mobile applications. *Advances in Engineering Software*, 40(12):1318 – 1328, 2009.

[43] Harri Oinas-Kukkonen and Virpi Kurkela. Develping successful mobile applications. *Internatinoal Conference on Computer Science and Technology (IASTED)*, pages 50–54, 2003.

[44] OMG Architecture Board ORMSC. Model Driven Architecture (MDA) Document number ormsc/2001-07-01, July 2001.

[45] openArchitectureWare. oAW Tutorial – Part I. Getting Started.

[46] Inés Pederiva, Jean Vanderdonckt, Sergio España, Ignacio Panach, and Oscar Pastor. The beautification process in model-driven engineering of user interfaces. In Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Barbosa, editors, *Human-Computer Interaction – INTERACT 2007*, volume 4662 of *Lecture Notes in Computer Science*, pages 411–425. Springer Berlin / Heidelberg, 2007.

[47] Paulo Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. In Philippe Palanque and Fabio Paternò, editors, *Interactive Systems Design, Specification, and Verification*, volume 1946 of *Lecture Notes in Computer Science*, pages 207–226. Springer Berlin / Heidelberg, 2001.

[48] Peter G. Polson, Clayton Lewis, John Rieman, and Cathleen Wharton. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, 36(5):741 – 773, 1992.

[49] Yvonne Rogers, Helen Sharp, and Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, third edition, 2011.

[50] Sabine Schröder and Martina Ziefle. Making a completely icon-based menu in mobile devices to become true: a user-centered design approach for its development. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, MobileHCI '08, pages 137–146, New York, NY, USA, 2008. ACM.

[51] Ben Shneiderman. *Designing the User Interface – Strategies for Effective Human-Computer Interaction*.

[52] Jared Spool and Will Schroeder. Testing web sites: five users is nowhere near enough. In *CHI '01 extended abstracts on Human factors in computing systems*, CHI EA '01, pages 285–286, New York, NY, USA, 2001. ACM.

[53] Adrian Stanciulescu, Jean Vanderdonckt, and Tom Mens. Colored graph transformation rules for model-driven engineering of multi-target systems. In *Proceedings of the third international workshop on Graph and model transformations*, GRaMoT '08, pages 37–44, New York, NY, USA, 2008. ACM.

[54] Jan Willem Streefkerk, Myra P. van Esch-Bussemakers, and Mark A. Neerincx. Field evaluation of a mobile location-based notification system for police officers. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, MobileHCI '08, pages 101–108, New York, NY, USA, 2008. ACM.

[55] Jean Vanderdonckt. A mda-compliant environment for developing user interfaces of information systems. In Oscar Pastor and João Falcão e Cunha, editors, *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 16–31. Springer Berlin / Heidelberg, 2005.

[56] Maiju Vuolle, Mari Tiainen, Titti Kallio, Teija Vainio, Minna Kulju, and Heli Wigelius. Developing a questionnaire for measuring mobile business service experience. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, MobileHCI '08, pages 53–62, New York, NY, USA, 2008. ACM.

[57] W3C Incubator Group. Model-Based UI XG Final Report, 2010.

[58] Yufei Yuan, Norm Archer, Catherine E. Connelly, and Wuping Zheng. Identifying the ideal fit between mobile work and mobile work support. *Information & Management*, 47(3):125 – 137, 2010.

Appendix A

Expert interviews

Expert interviews were conducted as the first step in data gathering in order to gain a better understanding of the current situation of mobile products already in the market, as well as the current user frustrations and application limitations. The expert interviews are covered in Section 4.1.2.

A.1 Interview questions

The interview was set-up to be *semi-structured*. The questions below were used to guide the interview, but the results are not limited to only these questions.

1. Do you mind if I make a audio recording of this interview? I use the recording as a supplement to my notes for the best result.
2. My goal: to understand points of improvement in the current application, and in particular with respect to the user interface.
3. With which recent project were you actively involved, and what was your role in that project? This concerns projects that involve a mobile application. (Max 2).
4. For each project:
   (a) Without involving the PDA, which tasks need to be conducted at the client side for this project?
   (b) Why are these tasks conducted?
   (c) For which tasks should the PDA offer support?
   (d) In which areas does the application support the user during his work?
   (e) In which areas does the application *disturbs* the user during his work?
   (f) What are, perhaps more generic, comments that you hear from users?

(g) Do you have any other comments you would like to say and are relevant to this interview?

5. Are there other people you would advice me to interview?
6. Do you have any comments regarding this interview?
7. Thank you for your time! I will write the results for this interview, and then validate those with you.

A.2 Results

A.2.1 Labels

The analysis of the interviews using open coding in grounded theory resulted in several labels to be assigned to independent data units. Table A.1 shows the labels that were identified by applying open coding in grounded theory during the analysis of the results.

Label	Description	#
HELPS	Data units with this label indicate that people are pleased with a mobile device that helps them during their jobs.	10
SW ISSUE	Data units with this label indicate that people experience software issues.	5
PROC POA	Data units with this label indicate that it is a point of attention that a mobile device forces a certain way of working.	4
HW ISSUE	Data units with this label indicate that people experience hardware issues.	3
HW PRO	Data units with this label indicate that people gain advantages by using mobile device hardware and its capabilities.	3
PRIV ISSUE	Data units with this label indicate that people have the feeling their privacy may be infringed.	2
INPUT ISSUE	Data units with this label indicate that people have issues inputting data.	2
MULTIMODAL PRO	Data units with this label indicate that the multimodal input (i.e., using device capabilities to help the user during input) is experienced to be pleasant.	2
PRES	Data units with this label indicate issues with the presentation of data on-screen.	1
SLOW	Data units with this label indicate that the device is experienced to be too slow.	1
PROC PRO	Data units with this label indicate that the mobile devices force a certain way of working, which is seen as a good thing.	1
SW FANCY	Data units with this label indicate that people are pleased with software features.	1

TABLE A.1: Labels, including their description and number of occurrences, that were derived from the expert interviews by applying open coding in grounded theory.

A.2.2 Data units

The results that are represented in the following table are aggregated results from all individual interviews.

Data unit	**Label**
No more paper, folders, forms, alarm codes, alarm addresses, directions, ... for workers to carry with them as they have been replaced by a device.	HELPS
The device supports a default style of reporting and a finite list of often-used options during reporting, making it less error-prone and easier to process	HELPS
Planning component is integrated.	HELPS
If a wireless internet connection is available, the device is able to show real-time and up-to-date supportive information	HELPS
The use of a PDA changes the way in which people work, and sometimes people have the idea they have to fill out more information using a PDA than before	PROC POA
The fact that a lot of data is registered on the PDA results in a feeling by users that they are being monitored.	PRIV ISSUE
People complain about the small size of the screen.	HW ISSUE
Devices come with a camera, but people complain about the quality of the pictures, which is lower than what they desire or expect.	HW ISSUE
In some branches, people want to work with a device that looks good, is sexy, and is comparable to popular consumer electronics.	HW ISSUE
Sometimes, non-relevant information is displayed on the device, which may not be a good practice given the small screen.	PRES
Touching the screen and making a selection of options is easy, but manual user input is often experienced to be difficult.	INPUT ISSUE
The devices act as a replacement of or support administrative tasks. Something that people using the device don't like spending time on.	HELPS
The availability of a camera is very desirable, as it allows for precise and detailed registration.	HW PRO

The collaboration between the device and a sophisticated third-party planning system makes not only the workers their jobs easier, but also that of the planners.	HELPS
As the PDA is also equipped with a GPS, it may also be used as a navigation device to guide the worker to a customer, eliminating yet another device.	HELPS, HW PRO
The PDA enforces a certain way of working, sometimes not compliant to the real-life way of working. In that case, workers put aside the PDA and use their own way, which results in erroneous data on both the device and backend.	PROC POA
Workers sometimes have the feeling that they are being checked on what they do because of the fact that everything is being registered.	PRIV ISSUE
Some user interface elements lack a “cancel” option.	SW ISSUE
The UI does not clearly show what information is required and what is optional.	SW ISSUE
Using a PDA enforces a uniform way of working. A great advantage for companies, but sometimes annoying to users, as it might not support their way of working.	HELPS
Historic data of a certain issue may be retrieved from a backend system, supporting solving an issue.	HELPS
Searching for items (can be really anything: locations, materials, issues, ...) should be supported by available means. In the case of locations for example by using a GPS, in order to filter the list from superfluous items.	MULTIMODAL PRO, HW PRO
The user interface is really tiny.	SW ISSUE
For some applications, it is near-to-enforced (because of choices in the UI, and small UI elements) to use a stylus, while users wish they didn’t have to. Some users however thinking the stylus is a great way of working. The problem with a stylus is that it is small and lost easily. In general, applications should be developed so that a stylus is not required.	INPUT ISSUE
Most applications provide poor feedback of what is going on, especially in case of errors. As can be seen with the public transport application, doing so significantly reduces the time required by the helpdesk to provide support.	SW ISSUE
After a software update, some sequences of interfaces may be changed that are not introduced, confusing the user upon first confrontation.	SW ISSUE

Users appreciate it when they are supported during input of information. For example, support scanning the barcode on a driver license so that the license number doesn't need to be filled out manually.	HELPS, MULTIMODAL PRO
Applications sometimes respond too slowly. A button is clicked, and nothing happens immediately on the screen. Because of this, users try to click again resulting in unexpected application behavior. At least some sort of indication that the application is busy should be provided. It cannot be that a user clicks the screen and that nothing happens immediately.	SLOW
The way of working enforced by a PDA increases the number of steps that must be completed before a task can be completed. On paper, people could skip steps that they could not fill out which would later be the problem of the person post-processing the information, instead of theirs.	PROC POA
Applications are very leading during the tasks of users (complete step 1, step 2, 3, ...). This is useful for new guys so that they know what do to. Experienced people go and perform their task, and then later put everything in the PDA as they aren't waiting for a device that keeps telling them what to do.	PROC POA, PROC PRO
Screen transitions with animations are very positively experienced by users.	SW FANCY

Appendix B

Field studies

Observations were conducted in the beginning of the design process, as they help to understand the user's context, tasks and goals [49]. The field studies are covered in Section 4.1.3.

B.1 Goals

The goal of this application is to identify the problem areas in the current version of the FMS-application, as used in the field. The result of this serves as input for a set of guidelines that will ultimately be developed so that the future application can deals with the problem areas.

B.2 Questions

In order to reach this goal, the following questions have been developed.

1. What tasks have to be supported by the PDA?
2. During what tasks does the user experience difficulties in terms of interaction with the device, and what are those difficulties?
3. What features of the PDA are experienced by the user as useful?

B.3 Method

The chosen method is a field study in the form of an observation, combined with informal interviews before, during and after the study. During field studies, observations can be conducted in order to see how users use the application during their daily work activities. This method allows to fill in details about the use of an application, that are more difficult to explain using interviews or questionnaires.

Framework. During the study, a framework is used to guide the observations. Although simple, the *who-where-what-framework* helps to keep goals and targets straight. Using this framework means that it will be identified who is being observed, what technology he or she is using, in what area, and what actions are performed.

Degree of participation. The observer will adopt a *passive* attitude so that the worker is as less bother during his work as possible.

Registration of data. During the field study some things will be written down at that moment. After the observation, the entire day will be reflected upon and experiences will be written down as soon as possible so that everything is still fresh in the observers memory. After that, this *mind-dump* can be processed, but at least the data has been registered.

Acceptance in the group. The goal is to improve the application that is being used by the persons being observed. for that reason, no problems are expected in this area.

Guaranteeing different perspectives. This observation is one from a series of observations, all at different locations. This way, a lot more information and also from different contexts of use will be obtained.

Think-aloud principle. Before the field study, the person being observed will be asked to be a bit more verbose than normal while using the device. This has the advantage that the observer knows what the observed is doing and what is going on in his or her head.

B.4 Practical issues

This field study may be conducted at less reachable or clean areas. It is therefor of importance to dress accordingly.

B.5 Ethical problems

The privacy of the worker will always be guaranteed. The identity will only be known to the observer and the person who scheduled the observation. This information is however not of importance for the results. In case the observed must be contacted after the observation, this will always happen through the observer.

Any confident statements, surroundings, systems, methods, information or otherwise obtained confidential information will be handled as such.

B.6 Results

Three field studies were conducted. The first one was conducted at a sewer and drainage cleaning company (in the table below denoted as *Sewer*) that uses the *Field Mobility Suite* for planning activities and registration of activities. A second field study was conducted at a municipality (*Municipality*), where parking attendants use the *CityControl* application by Sigmax to register and print fines for parking violations. The third and last field study was conducted with an inspector of railways (*Railways*), with whom we inspected several kilometers or railway and used the *FlexInspect* application to register issues.

B.6.1 Labels

Table B.1 shows the labels that were identified by applying open coding in grounded theory during the analysis of the results.

Label	Description	#
PRES ISSUE	Data units with this label indicate issues with the presentation of data or information in the application being too vague or unclear.	8
SLOW	Data units with this label indicate that people experience the device to be slow.	5
HW ISSUE	Data units with this label indicate that people experienced hardware issues.	4
INT POA	Data units with this label indicate interaction with the device is a point of attention.	3
SW ISSUE	Data units with this label indicate that people experienced software issues.	3
PROC POA	Data units with this label indicate that the device helps during people's jobs, but their work process must be taken into account. A point of attention.	3
HELPS	Data units with this label indicate that people are pleased with a mobile device that helps them during their jobs.	2
INT ISSUE	Data units with this label indicate that people experienced issues with device interaction.	1
QUALITY ISSUE	Data units with this label indicate that people experienced quality issues.	1

TABLE B.1: Labels, including their description and number of occurrences, that were derived from the field studies by applying open coding in grounded theory.

B.6.2 Data units

The results in the table below were constructed using axial coding in grounded theory, where the most important observations or claims by the observed persons were used as data units.

Context	Data unit	Label
Sewer	The list of planning assignments isn't sorted logically enough. Easier would be if the closest location would be selected.	PRES ISSUE
Sewer	De primary complaint is the device being slow or responding slow. The navigation in particular, it takes 1 minute before it boots up.	SLOW
Sewer	The stylus is primarily used as interaction mechanism.	INT POA
Sewer	A lot of scrolling through lists is required to select the correct item. In the case of RRS, drainage blockage often has the same reason. The application could note this.	PRES ISSUE
Sewer	The hardware keyboard is not easy to use because of the small keys on the keyboard. The use of it should be minimized.	INT ISSUE
Sewer	It was not clear that the device also supported a software keyboard.	INT ISSUE
Sewer	At first use, the use of the PDA was strange and it was unclear what fields were mandatory and what fields weren't.	PRES ISSUE
Sewer	The rubber bumper around the device is beginning to be worn off because the device is constantly put in and out of its holder in the car.	HW ISSUE
Sewer	At about 2/3rd of the screen from the left side a lot of scratches are visible. This is because at that location a lot of interaction is required to scroll through lists.	HW ISSUE, PRES ISSUE
Context	**Data unit**	**Label**
Municipality	Happy with the device! Works in most cases better than the papers that used to be filled out, and is convenient in its use.	HELPS
Municipality	Requesting information for a parking violation from the server takes a while (7 seconds). Unclear if this has to do with the PDA, data connection or the server.	SLOW
Municipality	The required information to see whether a parking permit is valid (something that is checked often), requires an extra user action.	PRES ISSUE
Municipality	In case of a violation, three pictures are always made. However, it takes a while before the camera starts up.	SLOW
Municipality	Sometimes, the application suddenly opens the fact book (an overview with all legal regulations). The reason is unclear.	SW ISSUE
Municipality	General complaint is that the device is slow.	SLOW
Municipality	The use is not kept up to date with what the device is doing. When the application is shut down, all data is synchronized with the backend system. This is a process that may take a while, but the user is not informed of the progress.	PRES ISSUE

Municipality	People have the feeling that the application gets slower after each update.	SLOW
Municipality	When the device is put in the cradle for charging, it switches to a wired connection. However, when the device was synchronizing, this process is aborted and not automatically resumed. As a result, this has to be done manually the next morning.	HW ISSUE, SW ISSUE
Municipality	When the device is removed from the cradle, it switches to a mobile data connection. Sometimes the device doesn't succeed in which case a popup is shown every 15 seconds.	HW ISSUE, SW ISSUE
Municipality	Only the stylus is used for interaction.	INT POA
Context	**Data unit**	**Label**
Railways	Rail inspectors have a huge document that describes all railway requirements. When changes are made to this document, it sometimes takes a while before the PDA is updated with the changed.	PROC POA
Railways	The grouping of information offers room for improvement. For example, there are multiple types of turnovers. Inspectors would like to select that they are inspecting a turnover, and then selecting the type.	PRES ISSUE
Railways	The number of inspection points per object differs per type. Not all questions are answered all time, because it are simply too many questions. More guidance could be made here, by requiring certain questions or placing important questions on top of the list.	PROC POA
Railways	A lot of scrolling is required through lists to select the desired item. In many cases the most used option could be placed at the top.	PRES ISSUE
Railways	The camera is used a lot. Its quality is fine, but a simple digital camera is used because of the better quality.	QUALITY ISSUE
Railways	With each inspections, inspectors can make a comment beforehand. This comment can be found back in the overview that is made available at a computer for processing. Some inspectors use this as a work-around to easily remember what a certain inspection was about.	PROC POA
Railways	Interaction is primarily done using a stylus. The hardware keyboard is only used as a trigger to make a picture.	INT POA
Railways	In general, inspectors are very happy with the device.	HELPS

Appendix C

Lab study

A lab study was conducted to investigate people's preferences with respect to browsing through hierarchical data structures on both smarpthones and tablets, using either more items per hierarchical levels, or less items per level but more hierarchical levels.

C.1 Basic design

Goals. Navigation through hierarchical menus or other forms of structured information is difficult [1, 28], especially on mobile devices. There is a balance between the number of hierarchical levels and the number of items presented on each level, and thus structuring hierarchical information is a challenge. Research shows that more hierarchical levels is better than more items per level and that each level ideally should have 4-8 items [28]. This research was however conducted using older mobile telephones, and an (for these days) outdated Windows Mobile devices.

Questions. There are two main question that we aim to answer by conducting this experiment:

1. Do users prefer a hierarchical setting with more items per level instead of a deeper drill-down in levels in order to select an item?
2. Is there a difference in this preference between a smart-phone and tablet setting?

Paradigms and techniques. This is a counterbalanced within-subjects design lab study. The order of the tasks will differ between groups. There will be four different participant groups, section C.3 elaborates on why. After each series of tasks the participant fills out a questionnaire about the task he or she just conducted. After the participant conducted the tasks, he or she is asked for a short interview. During the interview we will ask open questions, not answered in

the questionnaires, that elaborate on their experiences during the tasks. Video recordings will be made of the participants. There will be no statistical analysis done, but the questionnaires, interviews and video recordings will be used for qualitative analysis.

Section C.3 elaborates on the experiment setup.

Practical issues. It may be difficult to gather participants. And if they are gathered, it may be a challenge to find non-academic participants.

Ethical issues. A consent form should be read and signed so that each participant knows what he or she can expect, and we have the proper permissions to use the experiment data. Furthermore this experiment does not involve any activities that impose ethical issues.

Evaluation, interpretation and presentation. As mentioned, we expect that we may have some difficulties finding enough participants to conduct proper quantitative research. We therefor designed the experiment with focus on the qualitative part, and will be conducting manual analysis of the results. The results of this experiment will be used to improve a design guidelines document.

C.2 Participant tasks

The tasks that are to be conducted by the participant form an important part of this experiment. Each participant will conduct a total of 8 different tasks, spread over the different configurations. Every task will be conducted once by each participant.

Every task is associated with a difficulty rating. This rating is related to the depth of the hierarchy the user has to work through in order to reach the goal of the task. We denote difficulty with the variable **D**.

1. (D=7) Navigate to the **Zilverling Building**.
   *The Zilverling Building is located at the **University of Twente** in the **West** district of **Enschede**, which is located in the province **Overijssel** in **the Netherlands**, **West Europe**.*
2. (D=6) Navigate to **Google Headquarters**.
   *Google's headquarters is located in **Moutain View, Santa Clara County, California** in the **United State of America**, at the continent **North America**.*
3. (D=5) Navigate to **Plein van de Hemelse Vrede**.
   *The Plein van de Hemelse Vrede is located in **Peking**, in the **northern** part of **China**, a republic that is located in **Central Asia**.*
4. (D=7) Navigate to the **The Studio**.
   *The Studio is part of the **Sydney Opera House** in the **Northern** district of the **City of Sydney** in **Sydney**, which is located in the province of **New South Wales** in **Australia**.*

5. (D=7) Navigate to the **Euromast**.
   *The Euromast is in the **Centrum** of **Rotterdam**, located in the province **Zuid-Holland** in **the Netherlands, West Europe**.*

6. (D=6) Navigate to the **St. Basiliuskathedraal** (=Pokrovkathedraal).
   *The St. Basiliuskathedraal is located in **Moskou**, which can be found in the **Western** part of the **Russian Federation**, in **Eastern Europe**.*

7. (D=5) Navigate to **Machu Picchu**.
   *Machu Picchu is located in the **Cusco Region** of **Western Peru**, a country that is found on the continent of **South America**.*

8. (D=6) Navigate to **Mosselbaai**.
   *Mosselbaai is located in the district **Eden**, of the province **West-Kaap**, which is found in the country of **South Afrika**, in the **Southern** part of the continent **Africa**.*

C.3 Experiment setup

C.3.1 Independent variables

First of all, we will be using the same application on both a **smartphone** and a **tablet**. The layout will be different between these platforms, as we exploit the large screen that is available on the tablet device.

During all tasks we will be using the same data, but there is a difference in how it is structured. The **deep** representation uses a relatively large amount of hierarchical levels, but shows less items on each level. The **broad** representations uses a relatively small amount of hierarchical levels, but shows more items on each level.

Summarizing, we have four different configurations. 1: smartphone/broad (SP/B), 2: smartphone/deep (SP/D), 3: tablet/broad (TAB/B) and 4: tablet/deep (TAB/D).

C.3.2 Participant group/task distribution

We decided to create four participant groups. We do not alter between smart-phone and tablet to keep the number of groups feasible, and to make the experiment less difficult for participants.

In Table C.1 you can find the final distribution of tasks over participant groups. Every task is conducted by every group and on every device configuration. We aim to retrieve a multitude of 4 participants, with a minimum of 12.

C.3.3 Phased plan

Finally, we write down a phased plan for this experiment.

Task →	**1+2**	**3+4**	**5+6**	**7+8**
Group A	SP/D	SP/B	TAB/D	TAB/B
Group B	SP/B	SP/D	TAB/B	TAB/D
Group C	TAB/D	TAB/B	SP/D	SP/B
Group D	TAB/B	TAB/D	SP/B	SP/D

TABLE C.1: Groups/tasks distribution.

1. Welcome to experiment
2. Signing the consent form
3. Thank you for participating, explain what we'll be doing
4. Experiment starts
   (a) Participant fills out general questionnaire;
   (b) Participant conducts tasks according Table C.1, and fills out a digital questionnaire on a laptop after each configuration (or pair of tasks).
5. Participant is asked for a brief interview to elaborate on his or her experiences, as well can we ask questions about certain, characterizing, choices that the participant made throughout the experiment.
6. Participant gets a cookie

C.4 Consent form

Hello, and thank you for participating in this experiment! Your participation is very much appreciated. The experiment is part of the thesis that I, Mark Oude Veldhuis, am writing to obtain the degree of Master of Science in Human Media Interaction.

This experiment will take approximately 20-30 minutes of your time. You will be asked to conduct series of tasks on a smartphone and tablet. The exact details of the tasks will be disclosed when the experiments starts. During the experiment there will be a total of 4 moments where we ask you to fill out a short questionnaire about what you just did and what your experiences were, as well will we ask you to fill out a short questionnaire that provides some information about you. These questionnaires includes closed, multiple-choice and open questions. After you completed all tasks and filled out all questionnaires we may want to ask you a question or two about your experiences.

If at any time during the experiment you find yourself feeling uncomfortable or want to quit the experiment for whatever reason, please let us know.

The experiment is completely anonymous. A unique number will be attached to this session, but your identity cannot be obtained or derived from this number. Your name on this consent form will not be disclosed to anyone, and is only available to the experiment conductors. The data that is obtained during this experiment will be aggregated and analyzed together with

all the results of all other experiment sessions. A video-recording of your session will also be made, but the recorded audiovisual data will not be disclosed to anyone but the experiment conductors and is solely used to analyze the results of this experiment. This data will not be shared with anyone except for the experiment conductors and supervisors. The final conclusion of this experiment results in an advice that will be shared with Sigmax Mobile Solutions B.V., an Enschede-based company with whom we collaborate.

If you have any questions or concerns after the experiment you can contact the Human Media Interaction department of the University of Twente. Please address your message to either the primary experiment conductor (Mark), or the experiment supervisor (Betsy).

M. (Mark) Oude Veldhuis — m.oudeveldhuis@student.utwente.nl
dr. E.M.A.G. (Betsy) van Dijk — e.m.a.g.vandijk@utwente.nl

Once again, thank your for your participation. Please sign this form if you agree with the above stated and hand it over to the experiment conductors. Oh, and I almost forgot to mention that you'll get a little reward after the experiment :-)

Your name	**Signature**	**Today's date**	**City**
			Enschede

tick this box	Optional: I hereby grant the experiment conductors permission to use the audiovisual material gathered during this session for publishing purposes such as scientific research papers and presentations, but my face may not be recognizable.

C.5 Questionnaires

1. Questionnaire at the beginning of the experiment (step 4a).
   (a) What's your age?
   Answer: range-based list
   (b) What's your gender?
   Answer: male/female
   (c) What is the highest level of education you attended?
   Answers: vmbo, havo, vwo, gymnasium, mbo, hbo, wo, phd
   (d) What kind of education did you attend? (multiple options possible):
   Answers: alpha, beta, gamma
   (e) Smart-phone questions:
      i. Do you personally own a smart-phone?
      Answer: yes w/ touchscreen, yes w/o touchscreen, no
      ii. Do you use a smart-phone on a daily basis?
      Answers: yes w/ touchscreen, yes w/o touchscreen, no

iii. How extensive would you rate your daily smart-phone usage?
If answer to 1(e)ii is no, this question is not asked. Answer: scale 1-7

(f) Tablet questions:

i. Do you personally own a tablet?
Answer: yes, no

ii. Do you use a tablet on a daily basis?
Answers: yes, no

iii. How extensive would you rate your daily tablet usage?
If answer to 1(f)ii is no, this question is not asked. Answer: scale 1-7

(g) Thank you for your answers! You may now look at the tasks you have to conduct during this study. Good luck!

2. Questionnaires under step 4b concern how the user experienced the task in the setting at that moment.

(a) It was easy to complete the task.
Answer: scale 1-7 (1: completely disagree, 7: completely agree)

(b) I had to perform too many steps before I got to the item I wanted.
Answer: scale 1-7

(c) There were too many choices from which I had to choose at each step.
Answer: scale 1-7

(d) Briefly explain your answers and elaborate on your experiences.
Answer: open text

3. The interview at step 5 is semi-structured. We prepared a single question to ask the participant that will open up a short discussion/conversation:

(a) What configuration, smartphone/tablet in combination with deep/broad data, has your preference?

Appendix D

Acceptance expert interviews

Expert interviews with software developers and architects were conducted to assess the acceptance the Model-Driven Engineering environment that was developed.

D.1 Interview questions

This expert interview is semi-structured. There are certain questions we would like an answer to, but the interview allows discussion.

1. Do you mind if I make an audio recording of this interview?
2. My goal: find out whether expert software developers think the propose MDE environment would help to reduce development time, or what changes should be made.
3. Do you have any questions about MDE in general, or the proposed MDE environment?
4. Without taking the proposed MDE environment in mind:
   (a) Now that you know what MDE is and what an environment looks like: what advantages do you see in adopting an MDE environment?
   (b) And what disadvantages do you foresee?
   (c) For what goal do you think MDE is best applicable?
5. Looking at the proposed MDE environment:
   (a) What do you think of the source model, as it was presented?
   (b) What do you think of the separation of models into three levels?
   (c) Do you think the proposed MDE environment helps to reduce development time for Sigmax applications?
   (d) What changes should be made to the proposed MDE environment be altered in order to help reduce development time?

6. Do you have any other comments about adopting MDE, or the proposed MDE environment?

7. Thank you for your time!

D.2 Results

Table D.1 shows the results of the expert interviews after applying the selective reading approach and open coding in grounded theory. The *source*-header denotes the first occurrence of the data unit. The headings *1-4* shows which participant made the corresponding claim. The *total*-header shows the number of participants that made the claim. The Table is sorted descending on the number of people that made a claim.

+/-	Sentence/claim (data unit)	Source	1	2	3	4	Total
+	MDE is useful for quick proof of concepts or mockups	1	x	x	x	x	4
+	Proposed source model helps to structure applications	1	x	x	x	x	4
+	Proposed environment can help Sigmax if it is extended	1	x	x	x	x	4
!	The generated code should allow customization and extensibility	1	x	x	x	x	4
+	MDE brings consistency in development and makes it predictable	1	x	x		x	3
+	MDE prevents repetitive work	1	x	x	x		3
-	MDE may be difficult to adopt and get accepted in an existing development process	1	x	x		x	3
-	If the source model isn't good enough, developers miss freedom	2		x	x	x	3
+	MDE is useful for the development of multi-platform applications	1	x		x	x	3
+	Transformation separation helps to manage complexity	1	x	x	x		3
+	MDE allows for faster development.	2		x		x	2
+	MDE helps to focus on the design, you are not distracted by the details	2		x	x		2
+	MDE brings standardization	3			x	x	2
-	MDE uses a level of abstraction not easily understood by all	1	x			x	2
-	To setup an MDE environment may be difficult	2		x		x	2
-	Debugging the generated application becomes difficult	2		x	x		2
-	The adaptability or extensibility of generated code may be difficult	1	x	x			2

-	It is difficult to find the right level of abstraction for a source model	2		x	x		2
+	Proposed source model helps to develop multi-platform	1	x	x			2
-	Talking about screens means you have to know the target platform	1	x		x		2
!	Instead of generating for Android, you could generate so that a framework is used	2		x		x	2
-	MDE is less suitable for products	1	x				1
+	MDE is suitable for custom solutions	1	x				1
+	MDE helps to develop less error prone (eventually)	1	x				1
-	MDE introduces a huge learning curve for users and developers	1	x				1
-	MDE may be difficult to connect to other development tools	1	x				1
+	MDE is useful to automate development of techniques that are well specified	2		x			1
-	May be difficult to transform requirements to patterns in source model	1	x				1
+	The level of abstraction in the proposed source model is good	1	x				1
?	Better to splits from SigmaxApp to Screens based on the target platform	1	x				1
?	The SigmaxApp model will change over time (not a bad thing)	1	x				1
!	It has to be researched how MDE can be tailored into a development process	1	x				1
?	MDE may be better useful for the generation of parts of the entire application	2		x			1
!	A proper graphical representation of a source model is still required	2		x			1
-	Specific UI requirements should be able to be defined in the source model	3			x		1
-	User interfaces may be difficult to collect in patterns	3			x		1
!	Before such an approach is adopted, management has to be convinced of the investment	4				x	1

TABLE D.1: Results after applying the selective reading approach and open coding in grounded theory to the expert interview results.