DOMAIN-SPECIFIC MODELING AND CODE GENERATION FOR CROSS-PLATFORM MOBILE AND IOT-BASED APPLICATIONS

Doctoral Dissertation of:

Eric Umuhoza

Supervisor:

Prof. Marco Brambilla

Tutor:

Prof. Stefano Ceri

The Chair of the Doctoral Program:

Prof. Andrea Bonarini

2016 – XXIX
I would like to express my sincere gratitude to my advisor, Prof. Marco Brambilla for guiding me patiently from the beginning to the end of my Ph.D.

My gratitude goes also to the reviewers of my thesis: Prof. Ivano Malavolta and Prof. María Valeria de Castro. Their comments, insights and recommendation offered different points of view and contributed to improve the quality of my work. I am thankful to Prof. Jordi Cabot who provided me an opportunity to join his team as Visiting Scholar.

I am thankful to all the co-authors of my publications. This work is also yours. A special thank goes to Rodi Jolak for sharing with me the joy of success and the sadness of failure.

Finally, I would like to thank my family for the endless support and encouragement. A very special thank goes to my wife for supporting throughout writing this thesis and my life in general.
Abstract

Mobile devices constitute the most common computing device. This computing model has brought intense competition among hardware and software providers who are continuously introducing increasingly powerful mobile devices and innovative OSs into the market. In consequence, cross-platform and multi-device development has become a priority for software companies that want to reach the widest possible audience. However, developing an app for several platforms implies high costs and technical complexity. Currently, there are several frameworks implementing different methodologies for cross-platform application development. Nevertheless, these approaches still require manual programming, which yields to high risks of errors, inconsistencies, and inefficiencies. On the other side, with the advent of the Internet of Things era, homes, cities and almost everything is becoming smart. This technology is extending rapidly, but it still needs to solve many problems that arise when a technology requires being available at any time, in any place, for everyone and for any device. So far the development of front-end and user interfaces for IoT systems has not played a relevant role in research. On the contrary, user interfaces must be a key part in the IoT ecosystem because they can play a key role in acceptance of solutions by final adopters. This research proposes to face the challenges of mobile and IoT-based applications development by exploiting abstraction, modeling and code generation in the spirit of model-driven development engineering. The main contributions include modeling languages and design methodology for mobile and IoT-based applications; code generators for implementation phase; and a model-driven framework for user behavior analysis. In parallel to my research topic, I carried out different experiments to understand and propose solutions to some supposed barriers in the adoption of model-driven development approaches. In this context, I tackled the issues of modeling effort and modeling languages usability. Regarding the effective modeling effort in the process of modeling software systems, the experiment showed that more than 60% of effort is spent in designing. Thus, the fault of supposedly unproductive processes should not be blamed on modeling, but to the (anyhow necessary) effort devoted to thinking about the problem and identifying the solution. Furthermore, a research conducted in order to understand how the designed languages are actually used and how they fit the users’ need showed that available languages are either too complex with respect to user need or do not exactly fit the domain. This happens mainly because of the lack of right involvement of end-users in the language development process. I proposed a user-centered approach allowing the adaptation of existing modeling languages to the user needs through a language simplification process.
Riassunto

I dispositivi mobili stanno diventando lo strumento più utilizzato per consumare e produrre informazione. Questa evoluzione ha portato con sé una sentita concorrenza tra i produttori di tali dispositivi e i sviluppatori dei loro sistemi operativi che continuano a introdurre prodotti sempre più innovativi. Inoltre, non c'è una chiara dominanza sia tra i produttori dei dispositivi mobili sia tra i proprietari dei sistemi operativi. Pertanto, le aziende produttrici delle applicazioni mobili, che intendono coprire una grande partizione del mercato, sono costretti di produrre la stessa applicazione tante volte quante sono le piattaforme mobili. Lo sviluppo della stessa applicazione per diverse piattaforme comporta alti costi e aumenta la complessità tecnica dovuta alla diversità delle piattaforme mobili. Esistono diversi approcci che implementano diverse tecnologie per lo sviluppo multi-piattaforme. Tuttavia, questi approcci richiedono, in grande parte, la programmazione manuale che porta con sé il rischio di introdurre errori nel codice, incongruenze e inefficienze. Inoltre, con l’avvento dell’Internet-of-Things (IoT), diversi oggetti stanno diventando intelligenti. Questa tecnologia si sta estendendo rapidamente, ma ci sono ancora problemi da risolvere che sorgono quando una tecnologia richiede di essere accessibile in ogni momento, qualsiasi luogo, per tutti e per qualsiasi dispositivo. Finora, lo sviluppo delle interazioni con l’utente per questi sistemi non ha ancora attirato la giusta attenzione dei ricercatori. Questa ricerca si propone di affrontare le sfide dello sviluppo delle applicazioni mobili e delle applicazioni per l’IoT sfruttando l’astrazione, la modellazione e la generazione di codice nello spirito dello sviluppo basato su modelli. I contributi principali di questa tesi includono: i linguaggi di modellazione e la metodologia di progettazione per le applicazioni mobili e per IoT; i generatori di codice; e un framework, basato su modelli, per l’analisi del comportamento degli utenti dell’ applicazione. In parallelo al mio tema principale di ricerca, ho effettuato diversi esperimenti per capire e proporre soluzioni ad alcune barriere nell’adozione delle approcci di sviluppo basato su modelli. In questo contesto, ho affrontato il tema del carico di lavoro effettivo legato alle interazioni cogli strumenti di modellazione nel processo dello sviluppo basato su modelli. Inoltre, ho condotto una ricerca per capire come i linguaggi progettati vengono effettivamente utilizzati e come rispondono ai requisiti degli utenti finali. Questa ricerca ha mostrato che i linguaggi disponibili sono troppo complessi rispetto alle esigenze dell’utente, o non si adattano esattamente al dominio. Questo accade soprattutto a causa della mancanza di un giusto coinvolgimento degli utenti finali nel processo di sviluppo dei linguaggi. Ho proposto un approccio centrato sull’utente, per l’adattamento dei linguaggi di modellazione esistenti alle esigenze degli utenti attraverso un processo di semplificazione del linguaggio.
## Contents

1 Introduction and Problem Statement  
1.1 Introduction .......................... 1  
1.2 Problem Statement ..................... 3  
1.3 Vision .................................. 4  
1.4 Contribution  
   1.4.1 Understanding the Modeling Effort in Model-based Development Approaches .................. 5  
   1.4.2 Modeling Languages Design and Usability .................. 5  
   1.4.3 Mobile and IoT-based Application Design .................. 5  
   1.4.4 Implementation: Automatic Code Generation ............... 6  
   1.4.5 Application Monitoring: Model-driven User Behavior Analytics 6  
1.5 Thesis Organization ..................... 6  

2 Background  
2.1 The IoT: Concepts ...................... 7  
   2.1.1 Device or Thing .................... 7  
   2.1.2 Category .......................... 7  
   2.1.3 Terminal .......................... 8  
   2.1.4 External System ..................... 8  
   2.1.5 Communication ..................... 8  
   2.1.6 Intermediary ........................ 9  
2.2 UI Modeling  
   2.2.1 ViewElements ...................... 9  
   2.2.2 Events ............................ 10  
   2.2.3 Actions ............................ 10  

3 State of the Art  
3.1 Mobile development approaches .......... 11  
   3.1.1 The Native Approach ................ 11  
   3.1.2 The Cross Platform Approach ........ 12  
3.2 Model-Driven Development Approaches .... 14
## Contents

3.2.1 Approaches Allowing Executability .......................... 16
3.2.2 Other Concerns .............................................. 23
3.3 UI Modeling for IoT Applications .......................... 24
3.4 User Behavior Monitoring ................................. 25

4 Revealing the Effective Modeling Effort .......................... 27
4.1 Introduction .................................................. 28
4.2 Approach ..................................................... 28
4.3 Experiment .................................................. 30
4.3.1 Experiment Preparation ................................. 30
4.3.2 Experiment Execution .................................. 31
4.3.3 User Characteristics .................................. 32
4.4 Results ....................................................... 33
4.4.1 The Ratio Between Modeling Time and DT ............... 33
4.4.2 Impacts of Scenarios on DT ............................. 33
4.4.3 User Questionnaire ................................... 34
4.4.4 Discussion ................................................. 35
4.5 Treats to Validity ........................................... 36
4.5.1 Construct Validity ...................................... 37
4.5.2 External Validity ....................................... 38
4.5.3 Conclusion Validity ................................... 39
4.6 Conclusions .................................................. 39

5 Language Simplification ............................................. 41
5.1 Introduction .................................................. 41
5.2 Overview ...................................................... 43
5.3 Selection of Language Elements ................................
5.3.1 Task ........................................................ 45
5.3.2 Sequence .................................................. 45
5.3.3 Parallel Execution ...................................... 45
5.3.4 Parameters ............................................... 45
5.3.5 Conditional Execution .................................. 46
5.3.6 Loop ....................................................... 46
5.3.7 Events ...................................................... 46
5.4 Definition of Language Variants ............................. 46
5.5 Modeling of Use Cases by End-Users ......................... 48
5.5.1 Scenarios .................................................. 48
5.5.2 Use Cases Definition .................................... 48
5.5.3 Assigning Use Cases to Users .......................... 48
5.5.4 Modeling Sessions ..................................... 49
5.5.5 Monitoring ............................................... 51
5.5.6 Execution Data Analysis ............................... 52
5.5.7 Durations .................................................. 52
5.6 Questionnaire .................................................. 54
5.6.1 Scenarios difficulty ...................................... 54
5.6.2 Language variants difficulty ............................ 54
5.6.3 Cases difficulty .......................................... 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6.4 Variants deficiencies</td>
<td>55</td>
</tr>
<tr>
<td>5.7 Language Evaluation</td>
<td>55</td>
</tr>
<tr>
<td>5.7.1 Language Variants Evaluation</td>
<td>56</td>
</tr>
<tr>
<td>5.7.2 Variant One</td>
<td>56</td>
</tr>
<tr>
<td>5.7.3 Variant Two</td>
<td>56</td>
</tr>
<tr>
<td>5.7.4 Variant Three</td>
<td>56</td>
</tr>
<tr>
<td>5.7.5 Variant Four</td>
<td>57</td>
</tr>
<tr>
<td>5.7.6 Elements Evaluation</td>
<td>57</td>
</tr>
<tr>
<td>5.7.7 Language Variant Selection</td>
<td>59</td>
</tr>
<tr>
<td>5.8 Generalization</td>
<td>59</td>
</tr>
<tr>
<td>5.9 Conclusion</td>
<td>60</td>
</tr>
<tr>
<td>6 User Interaction Modeling</td>
<td>61</td>
</tr>
<tr>
<td>6.1 Mobile IFML</td>
<td>62</td>
</tr>
<tr>
<td>6.1.1 Mobile Containers and Components</td>
<td>62</td>
</tr>
<tr>
<td>6.1.2 Mobile Context</td>
<td>63</td>
</tr>
<tr>
<td>6.1.3 Mobile Events</td>
<td>64</td>
</tr>
<tr>
<td>6.1.4 Running Example</td>
<td>65</td>
</tr>
<tr>
<td>6.2 IoT Integration</td>
<td>70</td>
</tr>
<tr>
<td>6.2.1 IoT Events</td>
<td>71</td>
</tr>
<tr>
<td>6.2.2 IoT Actions</td>
<td>73</td>
</tr>
<tr>
<td>6.2.3 Modeling Example</td>
<td>75</td>
</tr>
<tr>
<td>6.3 Design Methodology</td>
<td>76</td>
</tr>
<tr>
<td>6.3.1 User Interaction Patterns</td>
<td>76</td>
</tr>
<tr>
<td>6.3.2 Data Synchronization Patterns</td>
<td>76</td>
</tr>
<tr>
<td>6.3.3 IoT Patterns</td>
<td>79</td>
</tr>
<tr>
<td>6.3.4 Running Example</td>
<td>84</td>
</tr>
<tr>
<td>6.3.5 Modeling Tool</td>
<td>85</td>
</tr>
<tr>
<td>7 Automatic Code Generation</td>
<td>87</td>
</tr>
<tr>
<td>7.1 Introduction</td>
<td>87</td>
</tr>
<tr>
<td>7.2 Applied Use Case</td>
<td>89</td>
</tr>
<tr>
<td>7.2.1 The Movies Manager App Running Example</td>
<td>89</td>
</tr>
<tr>
<td>7.3 Code Generators</td>
<td>91</td>
</tr>
<tr>
<td>7.3.1 PIM-to-NC: Generation of the Native Apps from PIM</td>
<td>91</td>
</tr>
<tr>
<td>7.3.2 PIM-to-PSM-to-NC: Native Apps from PIM through PSM</td>
<td>91</td>
</tr>
<tr>
<td>7.3.3 PSM-to-NC: Native Apps directly from the PSMs</td>
<td>92</td>
</tr>
<tr>
<td>7.3.4 PIM-to-CPC: Generation of Cross-Platform Apps from PIM through PSMs</td>
<td>93</td>
</tr>
<tr>
<td>7.3.5 PIM-to-FSM-to-CPC: Cross-Platform Apps from PIMs through FSMs</td>
<td>94</td>
</tr>
<tr>
<td>7.4 Discussion</td>
<td>94</td>
</tr>
<tr>
<td>7.4.1 Effort Analysis for the Different Approaches</td>
<td>94</td>
</tr>
<tr>
<td>7.4.2 Industrial Experience: The Case of WebRatio</td>
<td>95</td>
</tr>
<tr>
<td>7.5 Conclusion</td>
<td>96</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction and Problem Statement

This chapter introduces the work reported in this document. The chapter is organized as follows: Section 1.1 introduces the work in terms of relevance and challenges of mobile and IoT-based application development; Section 1.2 defines the problems that my research attempts to solve; My approach is introduced in Section 1.3; Section 1.4 introduces the main contributions of this thesis; While Section 1.5 presents the organization of the thesis.

1.1 Introduction

Nowadays, mobile devices constitute the most common computing device. A vast array of features has been incorporated into those devices to address the different demands of users spanning from games to serious business. Today mobile devices are as powerful as desktop computers in terms of their computing capabilities. This new computing model have brought intense competition and innovation among devices, OSs, and application providers.

According to Gartner, by 2016 more than 300 billion applications will be downloaded annually. The number of apps that are available in the online markets has reached unseen numbers. In fact, by July 2015, the Google Play store counted 1.6 million of available apps while Apple’s App Store counted 1.5 million [4]. In parallel with these numbers, the market also expects an increase in the number of global smart-phones users, which is expected to surpass 2 billion by 2016 [1] in comparison with 1.4 billion users estimated in 2013 [4]. From those statistics we can expect a healthy market of mobile apps that would be powered by a steady increase in the number of mobile device users, who as of today have, on average, 41 apps installed on their devices [2]. Furthermore, the motivation of the software development companies to continue producing more and better apps is supported by recent industry figures, according to which global mobile
Chapter 1. Introduction and Problem Statement

App revenues are projected to surpass 76.52 billion U.S. dollars in 2017. ABI research forecasts in 2018, app revenues will be worth 92 billion U.S dollars [89].

The mobile domain presents new challenges to software engineering. In addition to an increasing request of mobile applications, developers must provide applications that work on all platforms, at least on the most competitive ones (Android, iOS, and Windows Phone). Moreover, the dilemma between browser-based and native interfaces remains relevant and challenges the capacity of organizations to meet the demand for mobile applications.

IoT-based Applications

With the advent of the Internet of Things era, homes, cities and almost everything is becoming smart. Those smart objects can sense the physical world by obtaining data from sensors, affecting the sensed world by triggering actions using actuators, engage users by interacting with them whenever necessary, and process gathered data to provide useful information to us. This technology is extending very rapidly, but it still needs to solve many problems that arise when a technology requires to be available at any time, in any place, for everyone and for any device. Moreover, it is very likely that there will be viable use cases for sensor networks located in areas that pose challenges to connectivity. This presents a problem because it’s difficult to have the Internet of Things (IoT) if you do not have the Internet to connect the things to. One very viable solution is to use mobile devices in an opportunistic manner to serve as the IoT gateway. Our mobile device, which is packed with sensors and numerous wireless radios, helps us seamlessly communicate with various other objects.

An increasing number of physical objects are being connected to the Internet at an unprecedented rate realizing the idea of the Internet of Things. Examples of such objects include thermostats and Heating, Ventilation, and Air Conditioning (HVAC) monitoring and control systems that enable smart homes. There are also other domains and environments in which the IoT can play a remarkable role and improve the quality of our lives. These applications include: transportation, health care, industrial automation, and emergency response to natural and man-made disasters where human decision making is difficult. Figure 1.1 shows the projected market share of dominant IoT applications. Moreover, the IoT offers a great market opportunity for equipment manufacturers, Internet service providers and application developers. The IoT smart objects are expected to reach 212 billion entities deployed globally by the end of 2020 [52]. Overall, the annual economic impact generated by the IoT is estimated to be in range of $2.7 trillion to $6.2 trillion by 2025 [45].

Currently, the IoT vision is mainly focused on the technological and infrastructure aspect, as well as on the management and analysis of the huge amount of generated data. In particular, so far the development of front-end and user interfaces for IoT systems has not played a relevant role in research. On the contrary, user interfaces must be a key part in the IoT ecosystem because they can play a key role in acceptance of solutions by final adopters. The intelligent things connected together can cooperate and exchange information, but their ultimate goal is to provide value to people. Such value can be perceived only through appropriate user interfaces, which visualize information (through dash-boards, reports, or infographics), let user navigate the information, and also interact with the sensors and devices, by setting properties, collecting data and regulating their
1.2 Problem Statement

Cross-platform and multi-device development is a barrier for today’s IT solution providers, especially SMEs, due to the high cost and technical complexity of targeting development to a wide spectrum of devices, which differ in format, interaction paradigm, and software architecture. The challenges of mobile and IoT-based applications development that my research attempt to solve can be summarized as follows:

- **Multi-Platform.** The market of mobile operating systems is fragmented and rapidly changing. The diversity of OSs available on the market oblige software developers that want to reach a large audience of users to develop their apps for each platform (at least for the most competitive ones such as Android, iOS, and Windows Phone [58, 88]) separately.

- **Different Front End Requirements.** From the user interactions perspective, mobile apps are expected to support a wider set of interactions that are captured by means of a tactile surface (interaction through a set of gestures like taps and swipes) and through the different sensors that are packed into the device (sensor-based interactions like rotate and shake). Moreover, front-end design of mobile apps must consider the size constraints imposed by the characteristics of the screens of modern mobile devices. In addition, the mobile apps must adapt to changes of the context (the communication network, the battery level of the device and the environment surrounding the user) to deliver the most efficient interface [27].

- **Resource Scarcity.** Even though the mobile OSs provide the optimization strategies...
to cope with the scarcity of resources on mobile devices (like memory and storage, battery, and the instability and diversity of the communication networks), mobile apps need to be able to receive system notifications (such as the battery level and new networks availability) and react appropriately to them in order to provide a consistent and reliable user experience.

- **Device Diversity.** Currently, the market offers several families of mobile devices such as tablets, smart-phones and emerging smart-watches. Each of them can show different amounts of information, uses particular navigation patterns and has a diverse set of sensors available at run-time. These differences imply that software developers have to create applications that can either adapt to the specific device in which they are running, or create different versions of the application, each of them targeting a specific device family.

- **Interaction with IoT Devices.** In addition to the above described issues of user interaction with mobile applications, this vision of the Internet of Things which sees mobile device as the IoT gateway obliges applications developers to consider the interactions between the *terminal*, mobile device in our case, and smart-objects.

### 1.3 Vision

Currently, there are several frameworks implementing different methodologies for cross platform application development (Web, Hybrid, Interpreted and Cross Compiled): examples include PhoneGap / Cordova, Appcelerator Titanium, and Xamarin. However, these approaches still require manual programming. The advantages offered by those cross platform approaches can increase if combined with Model-Driven Engineering techniques. So far, several authors have applied model-driven development (MDD) approach to the development of various aspects of mobile and IoT-based applications [102]. MDD is a development paradigm that uses models as the primary artifact of the development process, with the aim of supporting as many phases as possible, covering also executability, i.e., the possibility of getting executable applications out of modeling efforts. Usually, in MDD the implementation is (semi) automatically generated from the models which allows for gains in productivity and quality of the software to be built [23].

This research proposes to face the challenges of the mobile revolution by exploiting abstraction, modeling and code generation. Different MDD approaches for cross-platform mobile and IoT-based applications development are studied with the aim of providing a framework allowing applications developers to choose a MDD approach that meets their requirements.

To deal with the issue of device diversity, I propose a development strategy based on the following assumption: the characteristics of different devices of the same family (tablet, smart-phone, smart-watch, etc) do not change drastically. For instance different smart-watches are assumed to have more or less the same screen dimensions and different smart-phones are assumed to have roughly the same sensors. The proposed approach relies on a set of model-to-model transformations (M2M) that are applied over the same general model of the application. Each of these transformations will produce a new model that describes the shape the application will have in a particular device.
1.4. Contribution

family. E.g: when designing an application that should be used in a tablet, a smart-phone and a smart-watch, a software designer will first create a single general model of the application. Then, he will use a particular M2M transformation to generate a version of the model that is suitable for phones. He will then repeat the same process using different M2M transformations to obtain the model for the tablet and the watch. At the end of the day, the software designer will have four different models, that can be transformed into running code following various code generation strategies.

1.4 Contribution

The objective of this thesis is to provide model-driven development tools and methodology for developing the front-end of mobile and IoT-based applications. Before I jump to the main contributions of the major topic, I present, in the next paragraphs, the results of the researches conducted to better understand and propose some solutions to the supposed barriers to the adoption of model-based development approaches. In this context, the issue of modeling effort and modeling languages design have been addressed.

1.4.1 Understanding the Modeling Effort in Model-based Development Approaches.

Even though several studies provided evidence that the adoption of model-based approaches can significantly enhance both developers’ productivity and product quality, many software practitioners consider such approaches time-consuming, hence prefer to avoid using models which are believed as excessive, superfluous and unnecessary artifacts. As a result, software modeling is still not widely adopted in software development. I conducted a research with the aim of revealing whether it is modeling or designing that dominates the effort in the creative process of modeling software systems. The conducted experiments showed that more than 60% of the effort is spent in designing. This means that the fault of supposedly unproductive processes should not be blamed on modeling, but to the (anyhow necessary) time devoted to thinking about the problem and identifying the solution.

1.4.2 Modeling Languages Design and Usability.

I conducted a research [103] on modeling language usability to understand how the designed languages are actually used and how they fit the users’ need. The investigation showed that available languages are either too complex (e.g.: BPMN) with respect to user need or do not exactly fit the domain (e.g.: UML). This happens because of the lack of right involvement of end-users in the language development process. I proposed a user-centered approach allowing the adaptation of existing modeling languages to the user needs through a language simplification process.

1.4.3 Mobile and IoT-based Application Design

At design time, this research defined: (i) a mobile modeling language named Mobile IFML [27]. Mobile IFML is a platform independent modeling language designed for expressing the content, user interaction, and control behaviour of the front-end of mobile applications. It is an extension of a OMG standard called Interaction Flow
Chapter 1. Introduction and Problem Statement

Modeling Language (IFML) [24], designed to address the specific requirements of mobile devices; (ii) a IoT extension of Mobile IFML allowing the modeling of the interactions between the terminal (the device used to access the IoT system) and the IoT devices; and (iii) Design methodology—the modeling tool [3] and a set of design patterns allowing the modeling of recurrent problems in user interaction modeling for both mobile and IoT-based applications.

1.4.4 Implementation: Automatic Code Generation

This research [104] explored different approaches to generate the code for mobile applications from the models describing those apps. I conducted a comparative study to identify the best trade-off between different code generation strategies. This study showed that there is no approach better than others in absolute terms but provided some useful guidelines that can help to identify the best code generation strategy. In particular, the provided guidelines helped us to identify the best strategy for the WebRatio company.

1.4.5 Application Monitoring: Model-driven User Behavior Analytics

This research proposed a model-driven engineering approach that combines domain-specific languages used for the user interaction design of the applications, with further languages that describe different perspectives of the user tracking at runtime upon the resulting application. Combining those structural information with application usage data allows to enhance the understanding of how the users consume the provided content and thus help analysts and decision makers to take more informed decisions on the application design and evolution.

1.5 Thesis Organization

The reminder of this thesis is organized as follows: Chapter 2 presents the main concepts which characterize IoT-systems and recalls the main concepts of the IFML. Chapter 3 provides a detailed overview of the state of the art regarding mobile and IoT-based application development; Chapter 4 summarizes the experiment on understanding the effective effort devoted to modeling in model-based development; Chapter 5 describes the proposed approach towards the simplification and adaptation of modeling language; Chapter 6 presents the mobile modeling language and its IoT extension, and the design methodology; Chapter 7 presents different code generation strategies; Chapter 8 describes the proposed approach to user behavior analysis; and Chapter 9 summarizes the results obtained and proposes possible future research directions.
CHAPTER 2

Background

This chapter presents the main concepts that characterize an IoT System in Section 2.1 and Section 2.2 overviews the main concepts of the Interaction Flow Modeling Language (IFML), used as reference language to model the user interactions.

2.1 The IoT: Concepts

This section introduces the main concepts that characterize an IoT system.

2.1.1 Device or Thing

The term *device* we mean all types of devices which can generate information (about physical event or state) and initiate, modify, or maintain those events or states; or that can perform actions. Those devices can be uniquely identified and communicate with each other or with other applications.

2.1.2 Category

The IoT devices can be grouped into different *categories* based on some criterion such as type, features, and geographical location. The categorization of IoT devices has many advantages that include a simplification of the interactions with those devices. For instance, a user can send the same command to all devices of the same category in a single interaction instead of sending the same command to each single device, separately.
Chapter 2. Background

2.1.3 Terminal
A terminal is any device which can run an IoT application with a user interface which can control other devices through the network. The terminals gather information sent by other devices or stored in external systems and they present that information to the user through appropriate user interfaces. The most used terminals include desktop computers, laptops, smart phones, HMI displays, and smart watches.

2.1.4 External System
With external system we refer to all the systems, connected to a network, in which the information of the devices and terminals can be stored, processed and retrieved. A user can access, create, modify or delete data from an external system.

2.1.5 Communication
The devices can communicate in different ways and can be connected with terminals and external systems. Several communication protocols for the IoT have been proposed around the IEEE 802.15.X, a standard for lower power systems. Examples of the outstanding protocols for IoT include 6LoWPAN, Z-wave, and Zigbee. The directions of communication that can be established between terminals, external systems and devices can be two way or one way.

A two way communication is important when a confirmation is required to perform
some operations. The two way communication can be direct, Figure 2.2(a) or indirect, Figure 2.2(b), depending on whether it involves an intermediary or external element.

A one way communication is used in non-critical systems, when a confirmation is not required. The one way communication can be direct, Figure 2.3(a) or indirect, Figure 2.3(b), depending on whether it involves an intermediary or not.

2.1.6 Intermediary

In this work I consider both the devices which communicate directly with the application and the ones that require a gateway (referred to as intermediary). An intermediary can be a hub, a gateway, a raspberry, or any other device that acts as a connection point between the end point devices and the application. Figure 2.1 summarizes the interactions between the user and the IoT system. That communication is performed into two phases: (i) Interaction between the user and the terminal used to interact with the IoT system; and (ii) Interaction between the terminal and the rest of the IoT system (IoT devices, intermediary elements and the external systems). In this work I cover both phases of the user interaction with the IoT system.

2.2 UI Modeling

To describe how the users interact (using terminals) with IoT systems, I use the Interaction Flow Modeling Language (IFML). IFML is an OMG standard that supports the platform independent description of graphical user interfaces for applications accessed or deployed on such systems as desktop computers, laptop computers, mobile phones, and tablets. The focus of the description is on the structure and behavior of the application as perceived by the end user. The most important constructs of IFML are InteractionFlowElements, distinguished into ViewElements, Events, and Actions [25].

2.2.1 ViewElements

View elements are the elements of the user interface that display the content of the application. The ViewElements are further divided into: ViewContainers, the elements of the user interface which aggregate other view containers and view components; and ViewComponents, the elementary pieces of information constituting the interface.
Chapter 2. Background

Figure 2.4: IFML example: product search, listing and deletion. The model consists of a ViewContainer Products which contains two ViewComponents (Product Search form and ProductResultList list); and Product Deletion Action triggered once the user selects the Delete Event associated to ProductResultList.

2.2.2 Events

Event is the concept that represents the events that may affect the state of the user interface. The events can be produced by the user’s interactions, by the application, or by an external system. The effect of an event is represented by an interaction flow connection, which connects the event to the view container or component affected by the event.

2.2.3 Actions

The actions are placeholders for business logic, used to express the side-effects of the user’s interaction.

In IFML, the user’s interaction is described by putting together the interaction flow elements by means of the interaction flow connections. Figure 2.4 shows a simple example of IFML model where the user can search for a product by entering some criteria in the Product Search Form. The matching items are shown in a list. The selection of one item causes a delete action to be triggered on it. When the deletion is completed, the updated list of products is displayed again.
CHAPTER 3

State of the Art

This chapter describes the state of the art of model-driven development approaches applied to the development of mobile and IoT-based applications. The rest of this chapter is organized as follows: Section 3.1 presents different approaches to the development of mobile applications in a broad sense; Section 3.2 reviews the works that apply MDD approach to the development of various aspects of mobile applications; Section 3.3 contains MDD approaches applied to the development of the user interaction with IoT applications; While the state of art approaches that apply MDD to user behavior monitoring are reported in Section 3.4.

3.1 Mobile development approaches

The vastness and diversity of mobile devices and operating systems available on the market oblige companies to produce and deploy the same application several times, once for each of the different mobile platforms. In some cases, application vendors rely on multiple development teams, each of which specializes in developing applications for a specific operating system, in other cases they rely instead on one of the strategies of cross-platform development. Currently, the two different approaches to the development of applications for mobile devices are native approach and cross platform approach [33].

3.1.1 The Native Approach

The native mobile application development approach produces applications specifically designed to run optimally on a device’s operating system. The main benefit that comes from this approach is that applications have access to the hardware and software features installed on the device (NFC, accelerometer, GPS, camera, magnetometer sensor, proximity sensor, luminosity sensor, positioning sensors and so on).
Chapter 3. State of the Art

Another benefit of the native approach is the possibility to publish the application on the stores where users are used to search for the applications they want to install on their Smart-phones. A native application installed on the device, moreover, is also accessible off-line unlike what happens for the web applications that necessarily require the Internet connection. The native graphical interface is generally preferred to the one interpreted by the browser because is more reactive, fluid, and the users are already used to the widgets provided by the platform of their devices.

Providing a native implementation might not be always preferred due to the development cost, in terms of resources and time associated with the development activity for each platform. Going by native methodology the vendor has to create and maintain separate solutions for each platform. That because the native applications are by definition, implemented and compiled based on a given platform. The applications for iOS (iPhone, iPad and iPod), for example, are written in Objective-C, those for Android are developed in Java, while those for Windows Phone are developed in the .NET environment. Maintaining separate applications and corresponding code is definitely a pain point for the application vendors [33, 87].

3.1.2 The Cross Platform Approach

This approach aims on creating a single application which can be used across multiple platforms. An application can be made cross platform in different ways: Web approach, Hybrid approach, Interpreted approach and Cross Compilation approach. Each methodology has its own purpose and context. The next paragraphs describe each methodology providing its pros, cons and some available tools that support that cross platform methodology.

Web Approach

A mobile web application is an application designed to execute in the web browser of the mobile device. Mobile web applications are developed using web technologies such as HTML, CSS and JavaScript. In this approach the mobile device will not have any application specific components installed. The applications are browser based and the application data is server driven. Since the application is browser based, it is platform independent.

Advantages One of the main advantages of this approach results on the possibility to use standard web programming tools such as HTML, CSS and JavaScript. Furthermore, this approach does not require any installation in to the mobile device. The application can be accessed using a URL through the mobile web browser. Since the data and application are hosted on the server, any update to the application is maintenance-free on the device. Since the mobile web browsers are standardized, the Web User Interface can be reused across different platforms.

Challenges An important limitation of this approach is that the applications produced are limited in the access to the device capacities (e.g. sensors, camera) and have a user experience that is inferior to native applications. A web application might suffer from inferior performance due to connection and network delays. Applications cannot be distributed through mobile application stores; they have to be accessed using the URL.
3.1. Mobile development approaches

The absence of an application in the application store might have a negative impact on its popularity. Unlike desktop applications, a mobile web application has to support different screen resolutions. This is one of the important factors which need to be considered while developing mobile applications. The application developer has less control on how different browsers render the content.

AppsBuilder [9] and iBuildApp [10] are examples of tools that offer the possibility to create free web application without writing any line of code. These tools offer a graphical environment to build the views of the application, which are automatically created in HTML5, thus supporting an approach to cross-platform.

Hybrid Approach

Hybrid approach stands in between web and native methodology. The hybrid applications are developed using standard web technologies and are executed inside native container on the mobile device. Hybrid approach uses the browser engine of the device which renders and displays the HTML content in full screen Web view control. The device capabilities are exposed to the hybrid application through an abstraction layer that exposes these capabilities as JavaScript APIs. This approach can take the advantage of both browser engine and device capabilities. Hybrid approach can be used for both server backed and standalone applications. Unlike web applications, Hybrid applications need to be downloaded and installed on the mobile device. PhoneGap/Cordova [11], is the most used tool based on this approach [72].

Advantages  A hybrid application is distributable through the application store. The main advantage of hybrid approach is that the user interface can be reused across different platforms utilizing native platform features. Since native platform features are made available through the hardware abstraction layer, application can make use of device features. Thus the web application is powered by device computing capabilities.

Challenges  The hybrid applications are inferior in performance compared to the native applications since the execution is performed by the browser engine. Since a hybrid application uses JavaScript abstraction layer to access the device capabilities, it is subjected to cross space communication vulnerabilities. Hybrid applications also suffer from platform specific behavior of JavaScript and threading model incompatibilities with JavaScript. Even though the user interface can be reused across different platforms the user interface will lack the look and feel of native application. To achieve the native look and feel the platform specific styling might be required.

Interpreted Approach

The application code is deployed to the mobile device and gets interpreted thereafter. The interpreter interprets the source code at runtime across different platforms and thus supports cross platform application development. The interpreted application interacts with the abstraction layer to access the native APIs. Interpreted applications make use of platform-specific native user interface elements for user interaction. Thus with this approach it is possible to create a user interface that corresponds to a native one. The application logic is captured in a platform independent way such as a set of commands.
Chapter 3. State of the Art

in XML or another description language. Examples of tools based on this approach include Appcelerator Titanium [8].

**Advantages** An Interpreted application provides the look and feel of the native application. The business logic can be reused across different platforms. Interpreted application is distributable through the application store. The device hardware and platform features are wrapped with specific framework Application Programming Interface.

**Challenges** Reusing the user interface depends upon the framework level abstraction. The main disadvantage of interpreted application is that the development is dependent on the feature set provided by the selected framework. The performance of the application might degrade a little due to the run time interpretation of the code.

**Cross Compilation Approach**

The cross compilation approach consists in the implementation of mobile applications by using the common programming languages. The developers can write the source code in a common programming language and the cross compiler compiles the source code into particular native code by converting the source code into native binaries. Examples of tools based on this approach include Xamarin [7].

**Advantages** Cross compiled application provides all the features that the native application provides. A cross compiled application have access to the hardware and software features installed on the device. All native user interface components can be used. The performance is the main highlight of these applications.

**Challenges** The main disadvantage of cross compiled application is that the user interface cannot be reused. Also the platform specific features such as camera access, location services, local notifications, etc., cannot be reused. These features are platform specific and the way to access them varies from platform to platform. This approach would be appropriate for simple applications but for sophisticated applications cross compilation will be outweighed by native approach.

### 3.2 Model-Driven Development Approaches

Model-driven development (MDD) is a development paradigm that uses models as the primary artifact of the development process. Usually, in MDD the implementation is (semi) automatically generated from the models which allows for gains in productivity and quality of the software to be built. Model-driven architecture (MDA) is the particular vision of MDD proposed by the Object Management Group (OMG). Therefore, MDA can be regarded as a subset of MDD, where the modeling and transformation languages are standardized by OMG.

In MDD, the core concepts are: models and transformations (i.e., manipulation operations on models). Let’s see how all these concepts relate together, revisiting the famous equation from Niklaus Wirth:

\[
\text{Algorithms} + \text{Data Structures} = \text{Programs}
\]
3.2. Model-Driven Development Approaches

In MDD context, the simplest form of this equation would read as follows:

\[ \text{Models} + \text{Transformations} = \text{Software} \]

Both, models and transformations need to be expressed in some notation, which in MDD we call a *modeling language*. Typically, when following an MDD approach, the running application can be obtained through one or more model transformations that subsequently produce a more and more refined version of the software description (manually completed/refined when necessary), until an executable version of it is reached. This refinement process can have as many iterations as necessary, depending on the company and the product to develop but, conceptually, the OMG proposes a three-level classification.

1. Computation-Independent Model (CIM): The most abstract modeling level, which represents the context, requirements, and purpose of the solution without any binding to computational implications. It presents exactly what the solution is expected to do, but hides all the IT-related specifications, to remain independent of if and how a system will be (or currently is) implemented. The CIM is often referred to as a business model or domain model because it uses a vocabulary that is familiar to the subject matter experts (SMEs). In principle, parts of the CIM may not even map to a software-based implementation.

2. Platform-Independent Model (PIM): The level that describes the behavior and structure of the application, regardless of the implementation platform. Notice that the PIM is only for the part of the CIM that will be solved using a software-based solution and that refines it in terms of requirements for a software system. The PIM exhibits a sufficient degree of independence so as to enable its mapping to one or more concrete implementation platforms.

3. Platform-Specific Model (PSM): Even if it is not executed itself, this model must contain all required information regarding the behavior and structure of an application on a specific platform that developers may use to implement the executable code.

A set of mappings between each level and the subsequent one can be defined through model transformations. Typically, every CIM can map to different PIMs, which in turn can map to different PSMs. The number of necessary models and their abstraction levels are project dependent. Usually, in the last step, the final code is generated by means of a model-to-text transformation from the design models.

The most direct benefits of MDE can be summarized as: increase of communication effectiveness between the stakeholders and increase in the productivity of the development team thanks to the (partial) automation of the development process. As a side effect, this automation reduces also the number of defects in the final code that could be inadvertently introduced by the developers.

The sequel of this section reports on the works that apply MDD approaches to the development of mobile applications. Those approaches can be grouped into two clusters depending on whether they target the *executability* of mobile apps (i.e., produce executable code for the IoT-based applications) or not.
Chapter 3. State of the Art

3.2.1 Approaches Allowing Executability

This section covers only the approaches that apply model-driven techniques to the development of mobile applications, covering the phases of development down to model execution either in terms of code generation or model interpretation. Before we jump to those approaches, we present a set of criteria defined within this research to assess various MDD approaches to the development of mobile applications. For a better understanding, the list of criteria has been structured into perspectives: development process, aspects covered, techniques, and generated apps.

- **Development Process Phases.** The development process dimension considers the main phases of the development process of the application: requirements analysis, design, implementation, and testing. The aim is to check which phases of the process are directly covered by each analyzed approach.
  
  - *Requirements analysis.* This criterion examines whether the approach in question covers the requirements analysis phase.
  - *Design.* This criterion examines whether the approach in question covers the application design phase.
  - *Implementation.* This criterion examines whether the approach in question addresses the coding phase.
  - *Testing.* This criterion examines whether the approach in question covers the testing phase.

- **Covered Mobile App Aspects.** Aspects covered perspective sums up criteria relating to the layer of the app covered by the MDD approaches.

  - *Content.* This criterion examines whether the approach in question covers the aspects related to the logical structure and operation to the data managed by the application.
  - *Business logic.* This criterion examines whether the approach in question covers the aspects related to the internal behavior of the application.
  - *User interaction.* This criterion examines whether the approach in question addresses the aspects related to the interactions between user and the application.
  - *GUI.* This criterion examines whether the approach in question covers the graphical user interface of the application.

- **Model-Driven Development Techniques Applied.** This dimension regroups the criteria related to the model-driven techniques used, e.g. visual modeling, code generation, or model-to-model transformations.

  - *Modeling language.* This criterion assesses how the applications are modeled in the approach under exam. It indicates whether the approach uses a graphical and / or textual concrete syntax.
  - *Multilevel code generation.* This criterion examines whether the approach in question uses intermediate models (e.g. platform independent model which maps to different platform specific models from which the code is generated) to describe the app requirements or produces the finally code directly from the
3.2. Model-Driven Development Approaches

initial models (e.g. final code generated from a platform independent model). This implies the use of model-to-model (M2M) transformations too.

– Executability. This criterion examines how the approaches addresses executability, i.e., it determines whether the approach relies on code generation or on model interpretation to execute the running apps.

• Generated Apps Perspective. The generated apps dimension examines whether the approach generates native, hybrid (aka., cross-platform), or web apps. Native apps consist of applications developed for a specific target platform, using a programming language or framework provided by the platform itself (e.g., Objective C, or Java), and compiled as an executable software for that platform. Hybrid or cross-platform apps are designed and developed once and executed on multiple platforms, typically thanks to HTML-based code, that is wrapped inside some kind of Web browsing technology and delivered as executable applications. Finally, by web apps we mean actual web sites, developed on purpose to be consumed mainly from a mobile device (mobile-first development).

Approaches

The considered approaches are grouped into: researches approaches and commercial solutions.

Research Approaches

MD2

MD2 \cite{59} is a model-driven framework for cross-platform development of data-driven mobile apps. In MD2, the application is firstly described in a platform independent model through a textual DSL. Then, a code generator (one for each platform of interest) transforms the PIM into source code, the business logic of the app along with necessary files to implement the GUI, for the corresponding platform. The MD2 framework provides in addition a code generator which creates a server back-end based on the data model of the application.

MobML

MobML is a collaborative framework for the design and development of data-intensive mobile apps \cite{50}. The framework is composed of three components: modeling languages, synthesizers (code generators) and a collaboration tool. It offers four platform independent modeling languages each of which addressing a different concern of mobile application:

• **Navigation**, which describes a mobile app as a collection of views and a set of navigation flows;
• **UI**, which describes the graphical interface of the app as a collection of graphic elements, with some extensions to represent the components of a particular target platform;
• **Content**, which models the data managed by the app; and
• **Business logic**, allowing the modeling of the internal operations of the app, and the interactions that occurs between the user and the app.

The synthesizers receive as input one or more models (up to four) of the application and generate cross-platform applications based on Apache Cordova.

17
Chapter 3. State of the Art

**Mobile Multimodality Creator (MIMIC)**

Mobile Multimodality Creator (MIMIC) [44] is a model-driven framework that enables the modeling and automatic code generation of multimodal mobile applications. MIMIC relies on the Mobile MultiModality Modeling Language (M4L), a language based on use of state machines to model input and output multimodal mobile interfaces. The M4L and its graphical editor have been specified through Obeo Designer. The mobile interaction modalities supported by the framework include the tactile, speech, and proximity. The framework generates multimodal interfaces for Android, iPhone and Web.

**Applause**

Applause[2] is a toolkit for creating cross-platform mobile apps. It consists of a DSL to describe mobile apps and a set of code generators that use these models to generate native apps for iOS, Android, Windows Phone and Google App Engine. Applause2, the second version of the framework, is expected to cover all the aspects of a mobile app.

**JUSE4Android**

JUSE4Android [39] is a model-driven tool that allows the automatic generation of business information systems (BIS) for Android. The apps are specified through annotated UML class diagrams from which the running code is generated.

**Francese et al.**

Francese et al. [48] propose a model-driven approach for the development of multi-platform mobile apps based on finite-state machine. The proposed development environment provides a finite-state machine editor (an Eclipse plug-in) and a generator, which produces source code starting from the model (Data flow, control flow, and user interaction) of the mobile app. The business logic of the application is written in JS. The editor allows developers to call device native features accessed through PhoneGap.

**MAG: Mobile Apps Generator**

MAG is a model-driven development approach to generate mobile apps for multiple platforms. The MAG approach is based on UML. The application requirements are modeled through use case diagrams, UML class diagrams are used to model the structure of the app while UML state machine diagrams are used for behavioral modeling. The mobile domain specific concepts are included in the application models thanks to the mobile UML profile [105]. MAG allows the developer to automatically generate the business logic code of the app from those models while the GUI of the app is developed separately.

**Vaupel et al.**

Vaupel et al. [107] propose a modeling language and an infrastructure for model-driven development of mobile business apps that support the configuration of user roles variants. Following this approach, different user roles are not combined in one app but lead to several app variants that may be configured after code generation. An app model consists of a data model defining the underlying class structure, a GUI model containing the definition of pages and style settings for the graphical user interface, and a process model which defines the behavior facilities of an app in form of processes and tasks.

---

1. www.obeodesigner.com/
2. https://github.com/applause/applause
3.2. Model-Driven Development Approaches

Chi-Kien Diep et al. Chi-Kien Diep et al [43] propose an online model-driven IDE which provides developers with a platform-independent GUI design for mobile apps. This approach consists on visually designing the GUI of the application once, as an abstract model, and transforming it several times targeting different specific platforms.

Mobl Mobl [60] is a DSL based on standard web technologies for the development of mobile web applications. It integrates languages for user interface design, styling, data modeling, querying and application logic into a single, unified language.

Rule-Based Generation of Mobile User Interfaces (RUMO) RUMO [95] is a model-driven development framework for multi platform user interface generation. The UI development in RUMO starts with the creation of a platform independent model which describes the basic structure of the user interface. Then, a set of rules has to be defined through a rule DSL in order to introduce application specific constraints such as usability or guidelines to guard the generation of the targeted user interfaces. Once the rules have been successfully checked, the PIM is finally transformed into target specific user interface (UI for Android, iPhone or for Win Phone). The platform specific UI generation involves the template mechanism which uses the predefined templates. Each template file is responsible of creating the source code for the desired platform. Furthermore, RUMO allows the creation of different versions of the template files as to address the issue of different versions of the same platform.

WL++ WL++ is a model-driven code-generation framework for developing multi–platform mobile apps as clients to existing RESTful back–ends [99]. WL++ framework is based on IBM Work Light platform, Backbone3 and the WL++ app modeling plugin. WL++ inputs the specification of an existing back-end service in the form of the APIs it exposes or the schema of the resource it serves, and infers the data model of the application. Then the developer enriches that application model by adding the proper views and the navigations among them. The source files are finally generated from those models through a set of pre-defined templates.

AXIOM AXIOM [62] is a model-driven approach for the development of cross platform mobile apps. It uses Abstract Model Tree (AMT), a consistent model representation, as the basis for model transformations and code generation. In AXIOM, the requirements of the application are are firstly described in platform independent intent models (interaction and domain perspective) using AXIOM’s DSL. Those intent models are then enriched with structural decisions and refined with platform-specific elements during a multi-phase transformation process to produce the source code for native apps: from requirements models to platform independent model (PIM), from PIM to platform specific models (PSMs), and finally from PSMs to running code.

Commercial Solutions

Mendix App Platform The Mendix4 App Platform enables business users and developers to build and deploy multi-channel apps with a MDD platform. The core library is offered

3http://backbonejs.org
4www.mendix.com
Chapter 3. State of the Art

as a set of standard services in the platform while specific libraries required for an optimal UI and user experience are built once and then offered as reusable widgets. In Mendix App Platform, the apps are built by defining visual models for the various app components such as the domain model, user interactions and business logic. Those models are then executed in a runtime environment. In addition, the platform provides core capabilities for non-functional application requirements, such as performance, scalability and security. Mendix combines model-driven development, the support of native device functions with the integration of Adobe PhoneGap to generate cross platform, hybrid apps.

IBM Rational Rhapsody  IBM Rational Rhapsody integrates with Rational Team Concert to offer modeling capabilities for Android applications and the visual representation of the Android framework API that developers can reference from within Rational Rhapsody. The references are then generated into Java code to automate the manual coding task. Additionally, Rational Rhapsody can read the AndroidManifest.xml file to visualize activities, services, broadcast receivers, main activity and content provides specified for better understanding of the application. The Rational Rhapsody Debugger also enables runtime animation of the class diagrams created for an Android app.

WebRatio Mobile Platform  WebRatio Mobile Platform is a model-driven development tool for the development of mobile applications [27]. The tool is based on the mobile-extended version of IFML standard [12]. WebRatio Mobile provides three integrated environment: (i) the modeling environment allowing the specification of user interactions through IFML diagrams and application content model through UML or ER diagrams; (ii) the development environment for supporting the implementation of custom components; and (iii) the layout template and style design environment, which allows the customization of UI through HTML 5, CSS and JavaScript. The code generated by the tool consists of ready-to-deploy cross-platform mobile apps, based on the PhoneGap.

Appian Mobile  Appian Mobile is part of the integrated Appian BPM Suite designed for mobile applications following the write-once, deploy anywhere architecture. In Appian Mobile, application designers can simply drag-and-drop to design mobile process patterns in Appian’s Process Modeler. Using a graphical Business Process Modeling Notation (BPMN) modeler, even business users can model and orchestrate processes, define and update rules, create forms and enable them to render natively in mobile apps. The User Experience is powered by Appian Self-Assembling Interface Layer (SAIL) which allows designers to create a single user dynamic interface definition, then deploy to native mobile client applications on major device platforms and across major web browsers.

Table 3.1 summarizes the classification of both research approaches and commercial solutions according to the proposed classification schema.

Trends and Challenges  Starting from the performed analysis, this paragraph identifies the current trends and suggests an outlook on the future of the Mobile MDD field.

5http://www.ibm.com/developerworks/
6www.webratio.com
7www.appian.com
### 3.2. Model-Driven Development Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Process</th>
<th>Aspects</th>
<th>Techniques</th>
<th>Apps</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDD²</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MobML</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MIMIC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Applanse</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JUSE4Android</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Francese et al.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MAG</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Vaupel et al.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Chi-Kien Diep et al.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mob</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RUMO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>WL++</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AXIOM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Commercial Solutions</td>
<td>Mendix App Platform</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>IBM Rational Rhapsody</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>WebRatio Mobile Platform</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Appian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 3.1: Model-driven approaches to the development of mobile applications and classification of their main characteristics.

- **Code Generation Approaches.** The application code can be obtained from models by following mainly two different code generation strategies: multilevel and single level code generation. When following multilevel code generation, the application is firstly specified in a platform independent manner, then the PIM is transformed into different PSMs, one for each platform of interest, from which the running code is generated. While in the single level code generation strategy, the final code is obtained by skipping one or more levels of the model-driven architecture. The analyzed approaches showed a preference of single level code generation over multilevel code generation. Most of the approaches like (MDD², MobML, MIMIC, WebRatio, and Mobl) generate directly the code from platform independent models PIMs skipping the PSM level. Therefore, despite the complexity of mobile context and the number of possible target platforms, most of the approaches do not consider efficient to have some intermediate PSMs, which are seen as an excessive burden.

- **Development Process** All analyzed approaches apply MDD to Design and Implementation phases. Only 23% of the approaches covers the testing phase, and only 35% covers the requirements phase. However, it is interesting to note that 3 out of 4 analyzed commercial platforms cover the Requirements. This means that requirements analysis is deemed a very important phase for production. Overall, only 11% apply MDD to all phases of the development process.

- **Mobile App Aspects.** Some approaches cover only few aspects of mobile applica-
Chapter 3. State of the Art

User interactions and GUI are the most covered aspects (94%) while only 70% of surveyed approaches address content and business logic aspects. This is explained by the fact that the most crucial points of mobile applications are related to user interaction. Therefore, the approaches specifically target these aspects with custom modeling solutions. Viceversa, more consolidated aspects like content and business logic can be covered with traditional modeling languages and approaches (E.g., UML, BPM and so on).

- **Executability.** Model-driven development approaches are commonly used to generate the final code either for a single concern or for all aspects of a mobile application. In fact, 94% of considered approaches apply code generation: the running code is generated from the high level models, while only the 6% relies on model interpretation. When following model interpretation, a generic engine is implemented and the model is interpreted by that engine, thus, model interpretation does not requires to generate the code to create a working application from a model. The code generation is preferred to model execution mainly for the following reasons:
  - Code generation is easier to start with and allows reusing existing programming artifacts. The developers can start using code generation by turning existing code into templates and replacing parts of the code with tokens which will be replaced by model information;
  - The generated implementation is easier to understand. The generated code is produced in a standard programming language that any developer can understand, while for model interpretation one needs to understand the generic implementation of the interpreter and the semantics of the model;
  - A code generator is usually easier to maintain, debug, and track because it typically consists of rule-based transformations, while an interpreter has a generic and complex behavior to cover all the possible execution cases;
  - Code generation provides an additional check for errors since the generated code needs to be compiled.
  - Generated applications are typically more easily accepted and integrated within an enterprise setting, because the generated code can be aligned with the company standards.

- **Native, Cross-platform or Web Applications.** Web applications are the less popular solution in MDD for mobile. Adoption of native and cross-platform applications is supported among others by the following advantages [38, 63] with respect to mobile-accessible web applications:
  - **Availability.** Users can easily find and download apps of their choice from the app stores and marketplaces;
  - **Offline.** With respect to web sites, apps have the ability to run offline;
  - **Safety and security.** Apps have to get the approval of the app store they are intended for.

More precisely, the model-driven community is focusing its attention on native applications development. In fact, 82% of considered approaches target the development of native applications. The main reason of this is the perceived higher
3.2. Model-Driven Development Approaches

quality of the obtained applications, in terms of performance, usability, and capability of exploiting and integrating with the most advanced features of mobile devices. This trend could be very interesting for the software development companies, especially to the SMEs. Indeed SMEs with limited resources, are currently obliged to go down the hybrid route, and thus, loosing some of the advantages of native apps in order to reach a large audience at a sustainable cost. Adopting a MDD approach would allow these company to get the same benefit obtained through native development, but with the productivity of cross-platform development.

- **Cross Platform Development.** When following MDD, with code generation, cross platform can be reached either by providing a code generator for each of targeted platforms or by generating the code required by the cross platform tool (like PhoneGap, Appcelerator Titanium, and Xamarin) to produce cross platform apps [104]. Basically, those code generators receive in input the same model describing the application and produce the code for the corresponding platforms. More than 88% of surveyed approaches target the development of at least two platforms.

- **Lack of Standard Mobile Modeling Language.** The modeling language is a fundamental building block of each model-driven development approach. Almost each approach relies on its own domain-specific modeling language either defined from scratch [44] or from the existing standards [104,105]. However, no specific standard has been devised for the mobile domain. This means that the community looses all the advantages offered by the standards. This should be one direction to target in the near future.

### 3.2.2 Other Concerns

This section assesses the existing works that apply the model-driven development techniques to other concerns of mobile applications or that are not focusing their attention to executability. Those works can be divided into two different clusters. In the first cluster we encounter a large corpus of researches that address conceptual modeling of software applications. Among the ones mainly focusing on the Web we can cite: (i) The Web Modelling Language (WebML) [32], defined as a conceptual model for data-intensive Web applications and conceived as a high level, implementation-independent conceptual model accompanied by the associated design environment, called WebRatio [13]; (ii) W2000 (formerly HDM) [20] which introduced a notion of model-based design, clearly separating the activities of authoring in-the-large (hypertext schema design) and authoring in-the-small (page and content production); (iii) OO-HDM [96], a UML-based approach for modeling and implementing Web application interfaces; (iv) Araneus [73], a modeling proposal for Web applications that allows one to represent the hypertext organization, with nested relations and inter-page links; (v) Web Application Extension for UML (WAE) [36], a UML extension for describing Web application interfaces and the client-server interactions; (vi) WebDSL [55], a domain-specific language consisting of a core language with constructs to define entities, pages and business logic, plus extensions; (vii) OO-HMETHOD [54], based on UML interaction diagrams; (viii) Hera [108], a model-driven design approach and specification framework focusing on the development of context-dependent or personalized Web information.

---

9Gartner predicts that more than 50% of mobile apps deployed by 2016 will be hybrid.
system. Some researches apply model based approaches for multi-device user interface development. Among them we can cite: (i) TERESA (Transformation Environment for inteRactivE Systems representations) \cite{21}, based on a so-called One Model, Many Interfaces approach to support model based GUI development for multiple devices from the same ConcurTaskTree (CTT) model; (ii) MARIA \cite{82}, another approach based on CTT; (iii) MBUE (Model Based Useware Engineering); (iv) UsiXML (USer Interface eXtended Markup Language) \cite{106} and (v) Unified Communication Platform (UCP).

In the second cluster we find a collection of works that propose model-driven solutions to address non functional requirements of mobile applications. In this cluster we encounter works that apply model-driven techniques to automate mobile applications testing. Namely, MobiGUITAR (Mobile GUI Testing Framework) \cite{16} is a model-driven testing framework which uses the state machines to test Android apps. Also Ridene et al. \cite{90} proposed MATeL (Mobile Applications Testing Language), a DSL allowing the modeling of test scenarios. Other researches apply model-driven approaches to address the issues of power consumption \cite{64,100}. Thompson et al. \cite{101} developed a model-driven tool, SPOT (System Power Optimization Tool), which automates power consumption emulation code generation.

Few research works that use model-driven approaches only for the purpose of modeling mobile apps can be found. Mobile IFML \cite{27} is a PIM designed for expressing the content, user interaction, and control behavior of the front-end of mobile apps. Recently, a variant of Mobile IFML has been applied in the tool WebRatio Mobile Platform to model the user interactions of cross-platform mobile apps \cite{12,104}.

Currently, the hot debate in the research and the development community in general is whether to go down the native app route (developing mobile apps for a specific operating system) or cross-platform (developing apps that work across multiple platforms) \cite{71}. Several works conducted research studies to compare the different development platforms using metrics such as complexity and user experience. For instance, Mesfin et al. \cite{75} conducted a comparative evaluation of usability of cross-platform apps on the deployment platforms. They observed that the usability of crossword puzzle app developed with PhoneGap (for Android, Windows Phone, and BlackBerry) was unaffected when deployed on the respective native platforms. The research performed by Tor-Morten et al. \cite{56} reviewed the mobile app development challenges and compared the issues and limitations of mobile platforms. Heitkötter et al. \cite{58} evaluated the cross-platform development approaches for mobile applications. However, all these studies are comparing the different development platforms (native and cross-platform) at the programming level. This thesis complements this discussion by offering a classification of model-based approaches for both native and web-based mobile applications development.

### 3.3 UI Modeling for IoT Applications

This section presents the approaches that apply MDD to the development of IoT-based applications. Those approaches can be grouped into two groups.

In the first group we encounter corpus of works that target the executability, i.e., produce executable code for the IoT-based applications. Among them we can cite: (i) FRASAD (Framework for sensor application development) \cite{79}, a node-centric, multi-layered software architecture which aims of filling the gap between applications and
low-level systems of sensor nodes. It provides a rule-based programming model which allows to describe the local behaviors of the sensor node and a domain specific language for sensor-based applications modeling. The final application code is automatically generated from the initial models; (ii) Pankesh Patel and Damien Cassou [81] proposed a development methodology which consists on separating the IoT application development into different concerns: domain, functional, deployment, and platform. This separation allows stakeholders to deal with those concerns individually and reuse them. The framework integrates a set of modeling languages to specify each of which allowing to describe one of the above mentioned concerns of the IoT applications; (iii) Jon Whittle et al. [47] proposed a MDD approach to generate efficient communication APIs to exchange messages with and between resource-constrained devices. This approach is based on ThingML (things modeling language) [49]; (iv) Ferry Pramudianto et al. [84] proposed a MDD approach which focuses on the separation of domain modeling from technological implementations. The framework allows domain experts to construct domain models by composing virtual objects and linking them to the implementation technologies. It allows automatic generation of a prototype code from the domain models and manual refinement of it.

The second group includes works that apply MDD to other aspects of IoT applications. Among them we can mention a MDD approach for the analysis of IoT applications via simulation [29]. Prehofer and Chiarabini [85] compared the model-based and mashup approaches, considering tools and methodologies for the development of IoT applications. They used UML as a reference modeling language and Paraimpu [83] for the mashup concepts.

3.4 User Behavior Monitoring

At the best of my knowledge this work is the first one which attempts to combine the information coming from the application models with runtime Web logs of the application execution, at the purpose of deepening the understanding of how the users consume the provided content. The works I found closer to the proposed approach are: (i) a Web Quality Analyzer proposed by Fraternali et al. [51], a framework which integrates the design-time conceptual schemas of the application and the usage data collected at runtime to analyze the quality of website. Their approach focuses on quality analysis while ours studies the user interactions; (ii) in [92]. Salini et al. proposed an approach that exploits the data coming from web usage analytics of existing web applications to generate the mobile navigation model of their corresponding mobile applications; and (iii) a model-driven approach for tracking information concerning the user navigation over Web sites proposed by Marco Winckler and Florence Pontico [112].

Web Log Analysis Web Analytics is the measurement, collection, analysis and reporting of Internet data for the purposes of understanding and optimizing Web usage [30]. The Web analytics process relies on defining goals, defining metrics, collecting data, analyzing data and implementing changes [110]. So far, several tools which support that process have been proposed. Among them we can cite Google Analytics, Yahoo Web Analytics, Compete, and ClickTale. All those tools provide the most important metrics: Unique Visitors, Visits/Sessions and Page Views [30].
Chapter 3. State of the Art

However, those tools provide analysis mainly at web page level and they do not take into account the content of the visited pages. There exist some analytics tools, like Google Analytics [70], which provide a way to track the user interactions at content level but at a high cost of hard coding which yields to high risk of errors and maintainability of the web application. Furthermore, those event tracking mechanisms support a small subset of user interactions. The proposed approach combines the application models with runtime Web logs of the application execution, at the purpose of deepening the understanding of how the users consume the provided content and thus help analysts and decision makers to take more informed decisions on the Web site design and evolution.

The log analysis field in general, and Web log analysis in particular, has attracted the research community due to several advantages offered by the tracking the interactions of the users with an information access systems. Agosti et al. [15] have presented the current challenges and trends of this field.
CHAPTER 4

Revealing the Effective Modeling Effort

In parallel to the main topic of this thesis, I carried out different experiments in order to understand and propose solutions to the supposed barriers in the adoption of model-driven development approaches. In this context the issues of: (i) **modeling effort**. Many software practitioners consider model-driven development approach time-consuming, hence prefer to avoid using models which are believed as excessive, superfluous and unnecessary artifacts; and (ii) **modeling language complexity and usability** have been tackled. This chapter presents the results of the experiment conducted to understand whether it is modeling or designing that dominates the effort for creating UML model. The research on modeling language usability and the proposed language simplification approach are described Chapter 5.

The remaining of this chapter is organized as follows: Section 4.1 provides the background and introduces the experiment; Section 4.2 describes the followed approach; Section 4.3 illustrates the design of the experiment and details its operational phases; Section 4.4 reports the results of the experiment; The threats to validity are presented in Section 4.5; and Section 4.6 concludes.

---

1 The content of this chapter has been produced in collaboration with Rodi Jolak, Truong Ho-Quang, and Michel Chaudron affiliated to Joint Department of Computer Science and Engineering Chalmers University of Technology and Gothenburg University Gothenburg, Sweden.
Chapter 4. Revealing the Effective Modeling Effort

4.1 Introduction

Models have emerged in software engineering as a powerful tool to tackle complexity of system specifications. Indeed, modeling allows to address the description of software based on different levels of abstraction and from multiple perspectives, in order to accommodate the needs of communication and description of a variety of stakeholders. In fact, models help to describe, reason, predict and evaluate both software problems and solutions. Furthermore, they provide effective means for supporting the communication between stakeholders, and serve as specifications for implementation [97].

Several studies provided evidence that the adoption of model-driven approaches can significantly enhance both developer productivity and product quality [34, 61, 80]. However, many software practitioners consider such approaches time-consuming, hence prefer to avoid using models which are believed as excessive, superfluous and unnecessary artifacts. As a result, software modeling is still not widely adopted in software development.

This work assumes that modeling is not time-consuming per se: it is the designing of the solution (i.e. pondering and making the design decisions) that consumes most of the time. If this is the case, it means that the fault of supposedly unproductive processes should not be blamed on modeling, but to the (anyhow necessary) time devoted to thinking about the problem and identifying the solution. This study wants to reveal whether it is actually modeling or designing that dominates the effort in the creative process of modeling software systems.

In order to assess this, I run a set of experiments about the creation of models in response to a set of requirements: I measure the time required to make the initial modeling of a system, and then we recreate the same model again, simply by redrawing the already defined solution. The assumption here is that for the copying of the solution, a subject does not have to (re)do any design thinking, but only spend effort on entering a solution in a modeling tool. At the end I assess the time difference between the two activities. I run the experiment based on modeling examples done in UML. UML (Unified Modeling Language) is a standard visual language adopted by the OMG that provides means to design domain and solution models in a semi-formal and well documented manner. It is a urban myth that UML modeling is perceived by many developers as a waste of time without any tangible contribution to the ultimate software product. Moreover, UML is the most used language for software architecture description in industry [68]. Therefore, it can be considered as a perfect candidate for this study.

4.2 Approach

The aim of this study is to dissect design time and model drawing time in UML modeling. In particular, I focus on software models as represented through UML domain model class diagrams. I define two fundamental sub-activities of the modeling task:

- **Design-thinking** as the act of understanding, reasoning and thinking about a design solution to be specified. The time devoted to this activity is referred to as Design Time.

- **Drawing and tool interaction**, as the act of expressing a visual diagram in a given modeling notation, plus the time spent for the interaction with the modeling tool.
4.2. Approach

The time time devoted to this activity is named *Model Drawing Time*. The research goal is to determine the effort spent on these two activities. I run a two-phase experiment. The analysis method consists of asking a set of subjects to:

- \((\alpha)\): create a domain model UML class diagram that addresses the assignment. Subsequently,
- \((\beta)\): re-draw the same modeling solution, as a copy of the diagrams produced in phase \(\alpha\).

The phase \(\alpha\) consists of the two aforementioned sub-activities. To determine the design time, I calculate the difference between the total time spent for modeling the solution \(T(\alpha)\), and the time spent on simply (re)drawing this existing solution to the problem \(T(\beta)\). Therefore, the results are given by these two equations:

\[
\text{Design Time } DT = T(\alpha) - T(\beta) \\
\text{Model Drawing Time } MDT = T(\alpha) - DT = T(\beta)
\]

Based on these values, I can define the Design Time Percentage (DTP) as the ratio of the Design Time over the total modeling time \(T(\alpha)\):

\[
\text{Design Time Percentage } DTP = \frac{DT}{T(\alpha)}
\]

The hypothesis here is that, more probable than not, design-thinking and drawing do not happen as multi-tasked activities, hence could be separated. Indeed in cognitive science, *task switching* is recognized as an important element of multitasking. It deems multitasking as switching from one task to another in quick succession rather than the concurrent performance of two or more tasks \[76\].

As a co-founding factor I consider the quality of the produced models, which can be used as a weighting parameter over the purely temporal information collected from the logs. I wanted to know how well the models reflect the problem domain, because extremely bad (or rough) models could affect the time statistics. To grade the quality of the models, a rubric has been defined before running the experiment. The rubric consists of a 5-point scale (1 is very bad and 5 is very good). In advance of the actual grading a set of possible ideal solutions was discussed. The grading was done in two steps: first the assessors graded all models separately, second they discussed the differences in grading and gave the model the final mark.

The experiment was conducted at Polytechnic University of Milan. Three different modeling scenarios have been formulated. Every scenario describes a system to be designed. A mix of B.Sc. and M.Sc. software engineering students were randomly given two modeling scenarios. To create their models, students were asked to use the WebUML \[40\] modeling tool. WebUML is an online UML editor which allows the logging of the modeling activities. I collected the recorded log files for each subject and assignment, and I evaluated the quality of each model with the proposed 5-point scale. I also setup an online Questionnaire through which participants answered questions about their background, UML experience, tool usability, and assignments understandability.

Based on the collected results, the research objective has been addressed by defining and responding to the following research questions (RQ):

---

29
Chapter 4. Revealing the Effective Modeling Effort

RQ1: What is the ratio between modeling time and design-thinking time?
RQ2: Does the type/size of the scenario affects the design-thinking time?

4.3 Experiment

In order to answer to the research questions presented in Section 4.2 I conducted a modeling experiment in which 48 participants were asked to design a solution for a given scenario using the WebUML editor. In this section I describe how the experiment was designed and conducted.

4.3.1 Experiment Preparation

Scenarios Definition

To make the analysis use case independent, I have evaluated the design time over three scenarios with different level of complexity and details. Every scenario describes a simple system to be designed. Those scenarios are reported in the next paragraphs.

Scenario 1 - E-learning. The system is used by teachers, students and an administrator (who is also a teacher). One teacher is responsible for many courses. Consider that courses consist of many topics. Students can enroll into different courses. There is a news section within the system. Teachers add news for a specific course and students read them. Every course ends with an evaluation test, teachers create a test and students have to do it. Students get one of these grades: fail, pass, good, or very good.

Scenario 2 - CamScanner App. We want to develop a simple app allowing a user to scan documents from a mobile phone and store/share them. The user can scan a document. The digitized document can be accessed, modified, or shared with friends on social networks. Scanned documents can be composed of several pages and are assigned a name. Users can assign tags to the scanned documents and can enrich them by adding annotations (notes or comments) or a watermark. Two or more scanned documents can be merged together to form a single document. The new document contains the references to the old ones.

Scenario 3 - Criminal Investigations. We want to develop a software for management of investigations of a Police Department. The system must support detectives, archivists, and supervisors. The detectives assigned to a case have read access to the data of a case; archivists can create new cases or modify /add details to existing cases, by uploading various material (facts, photographs and other media related to the case). Each material is characterized by a title, a short description, author, and its archiving code. Furthermore, the material can be tagged with the names of the persons associated to them (with role: suspects, witnesses and victims). The materials can be connected between each other. Cases can be closed. Only supervisors can reopen closed cases.

[https://drive.google.com/folderview?id=0B-bsX5PDK2vIMTQQdkJHSEdET2M&usp=sharing](https://drive.google.com/folderview?id=0B-bsX5PDK2vIMTQQdkJHSEdET2M&usp=sharing)
[http://editor.models-db.com/](http://editor.models-db.com/)
4.3. Experiment

Scenario 1
(1,2) (1,3)
Scenario 2
(2,1) – (2,3)
Scenario 3
(3,1) (3,2) –

Table 4.1: Valid experiments set

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>–</td>
<td>(1,2)</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>(2,1)</td>
<td>–</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>(3,1)</td>
<td>(3,2) –</td>
</tr>
</tbody>
</table>

In addition, I have defined a test scenario, used at the beginning of the modeling sessions to explain to the users how the tool works and to let them get familiar with using this tool.

Scenario Test - Social Activities. We want to develop a system for supporting people in the management their personal activities and projects in collaboration with their friends. The system shall allow to: create a new project (e.g., a dinner); associate the participants to the project; define activities to be performed in the context of the project (like call friends, book the table, select menu, etc.); assign a responsible person to each activity.

Assigning scenarios to participants

Having three different scenarios, the strategy allowing to get more generalizable results is to assign all the three scenarios to each participants. But considering the average time required to complete the experiment with one scenario, assigning three scenarios to each participant was not feasible due to the limited time the students had at disposition for the experiment. Thus, we decided to assign two scenarios to each participant. The submission order of the two scenarios to each participant is important because in the modeling of the second scenario, the participant will apply the experience acquired in the first one. In order to overcome unintended effects and to have balanced experiments, all the possible combinations were studied considering the order of scenarios. For that purpose, I used the Graeco–Latin square theory [66, 103]. Table 4.1 shows the valid experiments. Where the pair (x,y) represents the scenarios to be assigned to the participant and the order of assignment. For example, (2,1) means that the participant will model scenarios 1 and 2 starting with scenario 2 followed by scenario 1. It is considered not valid, an experiment which involves the same scenario, e.g. the pair (1,1) is not valid.

4.3.2 Experiment Execution

In this phase, the participant model the assigned scenarios by using WebUML. The editor has a logging system that logs participants’ actions (such as the creation, modification, and deletion of an element) useful to derive quantitative data allowing to compare and evaluate the produced designs and the time spent interacting with the tool. The experiment was conducted following this procedure in four steps:

1. Introduction. First, I have introduced the modeling tool (WebUML editor) to the participant through a training session. Then I have introduced domain modeling. A domain model captures the most important types of objects in the context of the business. The domain model represents the things that exist or events that transpire in the business environment.
Chapter 4. Revealing the Effective Modeling Effort

2. Instruction. During this phase I have explained to the participants the procedure of the experiment and how to save and submit their designs. Then, a short test of 15 minutes took place under our supervision in order to get accustomed with the basic functionality of the tool. This was done using a test scenario, equal for all the participants.

3. Modeling assigned scenarios. Each participant had to model two scenarios. For each scenario, the modeling task was done into two separate phases:

   I. Modeling phase. This phase encompasses the reading of the assigned scenario, the design of a solution to it, and the modeling of the proposed solution – the expression of the designed solution in a modeling notation. During the modeling phase, the participant can switch anytime between the reading of the assigned scenario and the modeling of the solution. This phase ends with the submission of the produced model. When submitting, the modeling tool generates (locally) an image containing that model.

   II. Redraw phase. In the redraw phase – referred also as copy phase – the participant redraws the proposed solution by copying it from the image generated during the modeling phase. During this phase, I monitor that the participants are neither working on changing the designs nor working on other tasks – to ensure that they are only copying their designs from the images generated in the previous phase.

   At the end of each phase, the participant was asked to clear the canvas (create a clean canvas) in order to allow a separate logging of two phases. Once the first scenario was modeled, the participant could proceed with the second scenario in the same way described before.

4. Closure. At the end of the test, after the participants had submitted all models, they were asked to answer a questionnaire about their personal information, knowledge about UML modeling and the WebUML editor, and the ease of understanding the scenarios.

The experiment was performed in a controlled setting: the participants work on computers in a lab at Polytechnic University of Milan. There was a supervisor that walked around to monitor that the participants worked on the assignment and not on other tasks or distractions.

4.3.3 User Characteristics

The experiment involved 48 participants. Overall, the participants are not experts in software modeling in general nor have particular experience in using UML. In fact, only 2 users claim to have high expertise in software modeling, 10 have a moderate expertise while the remaining 25 have low expertise. 65% of participants used UML only in small academic projects, 22% have never used it before the Software Engineering course while 5% (the master students) have used it in several several academic projects.
4.4. Results

This section presents the results of 37 users since the experiments of 11 participants were not valid (8 participants worked concurrently on the first and second task while 3 participants had technical network problems which prevented us from receiving their data) and then removed from our data set.

I used the statistical package R [86] to perform all tests. We chose significance level at 0.05, which corresponds to a 95% confidence interval. The normality of DTP’s was checked using Shapiro-Wilk test [98] and the Anderson-Darling test [19]. Table 4.3 shows the p-values of the two tests. As in both tests, the p-values are greater than 0.05, it can be (statistically) assumed that the design time portion is normally distributed.

The variance of the DTP with respect to the quality of models for scenario 1, 2 and 3 are presented in Figure 4.1, Figure 4.2 and Figure 4.3 respectively. It seems that the variance of DTP with respect to the quality is minimum.

### 4.4.1 The Ratio Between Modeling Time and DT

I calculated the mean and the standard deviation of DTP for each modeling scenario, and did the likewise for the quality of models. The results are presented in Table 4.2. In average, the design time of all scenarios is 61.09%. Furthermore, the distributions of DTPs for each scenario can be seen in Figure 4.4.

**Table 4.2: Statistical results for all scenarios**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Participants</th>
<th>DTP</th>
<th>Quality</th>
<th>Spearman’s rho</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std. Dev.</td>
<td>Median</td>
<td>IQR</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>21</td>
<td>64.46</td>
<td>18.35</td>
<td>4</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>22</td>
<td>58.22</td>
<td>18.90</td>
<td>3</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>25</td>
<td>61.14</td>
<td>14.60</td>
<td>4</td>
</tr>
<tr>
<td>All Scenarios</td>
<td></td>
<td>61.09</td>
<td>17.38</td>
<td>4</td>
</tr>
</tbody>
</table>

Results for Q1: For all scenarios, the average design time is 61.09%. The rest of the effort (38.91%) is dedicated to drawing and tool interaction, i.e. model drawing time.

### 4.4.2 Impacts of Scenarios on DT

Statistically, the RQ2 can be interpret as: are there any significant difference between the mean value of DTP between the three scenarios. In order to answer the question, I applied an ANOVA test. The following two hypotheses were formed:

Null Hypothesis:

\[ H_0 = \text{There is no significant difference between DTP of the three scenarios} \]

Alternative Hypothesis:

\[ H_A = \text{There is significant difference between DTP of the three scenarios} \]
Chapter 4. Revealing the Effective Modeling Effort

Figure 4.1: DTP vs. model quality score of each student (Scenario 1)

Table 4.3: Normality test results of DTP

<table>
<thead>
<tr>
<th>Normality Test</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shapiro-Wilk</td>
<td>0.0998</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>0.0931</td>
</tr>
</tbody>
</table>

Having the data complying with the *normal distribution*, it is possible to apply the One-way ANOVA test. Table 4.4 shows the results in detail. Since $p = 0.486 > 0.05$, the null hypothesis cannot be rejected. In other words, the differences between mean value of DTP over the given three scenarios are not significant. I took a step further by comparing the differences in the DTP mean value between each pair of scenarios. Tukey’s comparisons has been used for that test. The results, as reported in Table 4.5, show that the difference in the mean value of DTP is not significant between every pair of the three scenarios.

Results for Q2: There is a statistical evidence to conclude that the DTP stays the same through different domain modeling tasks (i.e. they are different in topic, size, levels of detail).

4.4.3 User Questionnaire

This section summarizes the answers gathered from the 37 users involved in the experiments. The users’ feedback complement the results presented in previous subsections.

Perceived Complexity of Scenarios Overall, the participants were able to understand the assignments. Indeed, 87% of participants declared that the assignments were clear, 5%
4.4. Results

Figure 4.2: DTP vs. model quality score of each student (Scenario 2)

Table 4.5: Tukey’s comparison results

<table>
<thead>
<tr>
<th></th>
<th>diff</th>
<th>lwr</th>
<th>upr</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2 - S1</td>
<td>-6.24</td>
<td>-18.63</td>
<td>6.15</td>
<td>0.453</td>
</tr>
<tr>
<td>S3 - S1</td>
<td>-3.32</td>
<td>-16.09</td>
<td>9.45</td>
<td>0.807</td>
</tr>
<tr>
<td>S3 - S2</td>
<td>2.92</td>
<td>-9.31</td>
<td>15.15</td>
<td>0.835</td>
</tr>
</tbody>
</table>

were not able to respond while 8% claimed that the assignments were not clear to them.

Modeling Tool Usability The participants did not report serious usability problems related to the modeling tool, the WebUML editor. Only 1 participant claimed that the tool is difficult to use, 13 participants were neutral while the remaining found the tool easy to use.

4.4.4 Discussion

Overall, this study showed that a majority of time devoted for modeling is spent on thinking about the design, and not on drawing it in a modeling tool. This should dispel the myth that modeling is a waste of time because it may be a key factor that prevents organizations from adopting a model-based approach.

As a further step in discovering how design time affects the modeling processes, I compare activity transition graphs of the modeling process with those of the copy (re-drawing) process. The two transition graphs, i.e. Figure [4.5 and Figure [4.6] are built using the Disco tool [57]. Each node in the graphs represents one single activity (i.e. add, create, move, set, remove) and shows the total number of its occurrences (across all cases). The transition arch from a node A to a node B indicates how many times the activity B was sequentially performed after the execution of the activity A. On each transition arch, the total duration of the transition is also showed. For example in Figure [4.6] the transition arch from Create to Remove indicates that there were 40 removing activities after creating a diagram instance, therefore the transition took overall 552 seconds.
Chapter 4. Revealing the Effective Modeling Effort

Figure 4.3: DTP vs. model quality score of each student (Scenario 3)

From the graphs we can observe that compared to the modeling tasks, participants spent significantly less time and performed fewer activities during the copying tasks. This is reasonable, since the students do not have to think about modeling solution. Moreover, they could also learn from mistakes made during the previous task, the modeling phase. In fact, they made less mistakes: the number of REMOVE-actions during the copy tasks is in average the half of the number of REMOVE-actions performed during the modeling tasks. I took Move activities as another example for the existence of DT during modeling tasks. Indeed, the Move-action was the dominant. On the other hand during copying tasks, participants seemed to spend less time for layout. They focused more on Creation and Edition of diagram instances. Removing diagram components seemed to be a hard (time-consuming) decision. It took on average 11.22 seconds and 7.07 seconds during modeling and copying tasks, respectively.

Interestingly, the most time-consuming decision was to add new components to a design after removing other components from it. It takes significantly more time than any other activity in both the modeling and copying tasks. There are 56 cases where participants made that type of decision around 3 times during the modeling tasks, 24.2 seconds for each time. On the other hand, there were only 36 cases where participants made that decision 2 times per case in the copying phase – they had to think around 8.5 seconds to make the decision.

4.5 Treats to Validity

This section presents the threats to validity for this study. They can be categorized by following the validity terminology introduced by Wohlin et al. [113]. In this research, three types of threats to validity are identified: Construction Validity, External validity, and Conclusion Validity.
4.5. Treats to Validity

4.5.1 Construct Validity

To the best of my knowledge this research is the first to introduce the notions of design time and model drawing time. The proposed approach for measuring design time is coarse: I subtract model drawing time from the total creation time of a software model. It may be possible that participants performed other tasks than thinking about the design when they were not modeling. At the same time that may also be the case in real life industry settings and thus not change much of the percentage of modeling time.

During the experiment, I observed that a few participants had consulted external resources from Internet. One could claim that this might affect the calculation of the design time, as the students might be spending their time for some other activities out of design (e.g. search strategy). However, I argue that even when searching for available solutions or suggestions, students had to make design decision, which is definitively one kind of design thinking.

I benefited from the fact that I performed the experiment in a controlled environment (in stead of as a homework assignment): 8 participants created the model and its copy in parallel. I eliminated these cases from the data set because I could not explicitly calculate the modeling and copying time. For replication of this research, a lesson learned is to instruct participants not to do the tasks (α and β) simultaneously.

I did not aim at maximizing realism and focused on class diagrams for various reasons: I wanted to ask simple tasks based on a well-known notation; I wanted to reduce confounding factors and thus keep more control over the experiment compared to drawing multiple types of diagrams; and I wanted to have a preliminary result that could validate my vision.

In the redrawing of a copy of the previously created solution, participants in the study

Figure 4.4: DTP distribution of each scenario
4.5.2 External Validity

Complexity of the scenarios. The scenarios were kept simple and clear so that the participants can easily understand and complete the tasks in the time of the experiment. In industry settings, modeling tasks can be much more complicated in term of size, terminology, languages, level of details, etc. which I could not cover in this study. This could somehow reduce the generalizability of the findings of this study when it comes to real-world cases. However, I consider this threat as acceptable for this preliminary investigation.

Participants and their modeling expertise. The participants being involved in the experiment may not represent the general population of modeling practitioners. Moreover, their modeling expertise are relatively homogeneous. This limits me from generalizing the findings to other subjects (i.e. experts, professional software architects, industrial practitioners in the field). I consider the findings as a basis to extend this study to larger community of modeling practitioners rather than a considerable threat to validity.
4.6 Conclusions

Figure 4.6: Transition graph between activities being performed during Copy Tasks

4.5.3 Conclusion Validity

I have performed statistical analysis for which I checked suitability (normality of the distribution of the data). I have thought about possible additional confounding factors, but could not think of any that could have a major impact on the measurements.

4.6 Conclusions

In order to better understand the effort involved in using software models in software development, the distinction between design time and modeling time has been introduced. Subsequently, I defined and ran an experiment in which I measure how much time (effort) each of these activities takes – both in absolute time and as a percentage of the total time spent on creating domain level class diagrams in a simple student assignment. From this experiment I conclude that the time spent on thinking about synthesizing the design takes around 60% of the time compared to 40% of the time that is spent on modeling the design in a UML modeling tool.

One implication of this research is that projects that omit modeling, run the risk of also failing to spend a healthy amount of time on thinking about the design. This suspicion aligns with an earlier finding that developers report that creating design models in the early stage of a software development projects, leads to better modularity of the design [80]. Moreover, the experiment showed that novice students were able to create models of a small system in a short time (less than half an hour). It is very unlikely that the effort required for creating models of larger system grows very rapidly, hence also there the effort for creating models is modest or even small compared to the time used for making the implementation of the software. From this perspective it seems more a matter of integrating the creation and maintenance of models in modern, mostly agile, software processes – e.g. through including creation/updates of a model into the definition of done.
CHAPTER 5

Language Simplification

The adaptation, specially by means of a simplification process, of modeling languages is a common practice due to the overwhelming complexity of most standard languages (like UML or BPMN), not needed for typical usage scenarios while at the same time companies don’t want to go to the extremes of defining a brand new domain specific language. Unfortunately, there is a lack of examples of such simplification experiences that can be used as a reference for future projects. This section reports on a field study aimed at the simplification of a business process modeling language (namely, BPMN) for making it suitable to end users. The proposed simplification process relies on a set of steps that encompass the selection of the language elements to simplify, generation of a set of language variants for them, measurement of effectiveness of the variants through user modeling sessions and extraction of quantitative and qualitative data for guiding the selection of the best language refinement. I describe the experimental setting, the output of the various steps of the analysis, and the results I obtained from users. Finally, I conclude with an outlook towards the generalization of the approach and consolidation of a language simplification method.

5.1 Introduction

In recent years, the increased adoption of domain specific languages (DSLs) and of general purpose modeling languages by practitioners with different role and expertise has paved the way to the need of providing ways to simplify such languages to adapt to the expectations of practitioners.

Indeed, in many cases adopters complain about complexity or impreciseness of notations, when facing the challenge of starting using new languages. In some cases, the language they are asked to use is not exactly matching the domain or need, while in
Chapter 5. Language Simplification

Figure 5.1: Empirical study on simplification of BPMN: At the beginning of the process, we select (the language designers) the BPMN elements to simplify; we then generate the language variants allowing us to assess the effectiveness of the language through the modeling sessions performed by end-users, in the third phase. In the fourth phase, we collect users’ feedback through a Questionnaire while in the last phase we evaluate the language variants based on the data gathered in previous phases. The outcome is a simplified BPMN, suitable for end-users.

Other cases the language complexity in terms of notation (number of symbols and syntax rules) or semantics, is overwhelming with respect to what is actually needed.

Therefore adaptation of modeling languages, especially in terms of simplification, has become a common need in companies. This avoids creating of completely new domain specific languages [94] or further increasing the size of existing language to match the specific needs (e.g., through profile-like mechanisms), without dropping useless parts [69]. Unfortunately, neither precise processes nor practices for addressing this problem are studied, and very few examples of language simplification experiences are reported [5]. In existing experiences users are often involved in the initial phases of the process [65], are asked to provide examples of models [35][93], or to setup discussion sessions [31], but users rarely take an active role along the whole language definition process [65][74].

This section reports on a field study aimed at the simplification of a domain-specific modeling language by means of direct involvement of users through quantitative and qualitative analysis of their modeling experience. In this specific case, I consider the problem of simplifying a standard business process modeling language for making it suitable to end users. General purpose business process modeling languages are widely known and adopted. Despite their expressive power and notations vary a lot, they share some criticisms about their overwhelming complexity with respect to the typical usage scenarios and intended adopters. The most prominent example is the Business Process Modeling Notation (BPMN). It offers a vast range of modeling constructs which turns it in an overly complex language [91][67], which makes it an under-used language in most situations and by most practitioners. Indeed, business analysts frequently use
arbitrary subsets of BPMN. Michael zur Muehlen and Jan Recker [78] raised the question "How much language is enough?" and evaluated quantitatively which of the modeling constructs provided by BPMN are used regularly. Their findings showed that less than 20% of BPMN constructs are used regularly. However, the usability study alone is not enough to determine which fraction of the language should be preserved or removed, since this depends a lot on the intended use and therefore it might not be enough to rely on the most used elements only.

Thus, simplification is not an easy process, and needs to be applied carefully, considering the concrete objectives of the simplification. The case of personal process management has been studied. Personal process management refers to the application of BPM techniques and tools to personal task management, to-do lists, and small work plans shared with friends and personal contacts. The introduction of the concept of process and execution flow in personal, everyday life tasks could allow users to manage their activities in a more structured, coherent and consistent way. We call this Personal Process Management (PPM) [22]. The language for modeling such processes should be complete enough for describing basic processes but also simple enough to let people understand, accept and use them in their everyday life [111].

The proposed simplification process relies on a set of steps that encompass: (i) the selection of the language elements to simplify; (ii) definition of a set of language variants for those elements; (iii) measurement of effectiveness of the variants through modeling sessions performed by end-users and explicit questionnaires; and (iv) extraction of quantitative and qualitative data for guiding the selection of the best language refinement. Notice that in this work we do not focus on the graphical notation to be used (aka., concrete syntax, that is the graphical symbols to be adopted), but only on the expressive power of the language.

5.2 Overview

This section describes the study on simplification of BPMN with the aim to make a suitable language for personal, every life tasks modeling. In this work, I use Personal Process Modeling Language or equivalently, a Notation for Business Process Modeling to refer to that simplified version of BPMN. The simplification process relies on the following phases (Figure 5.1):

1. Selection of BPMN elements to be simplified.

   In this phase, I identify the elements of the BPMN that are suitable for the modeling of personal processes. During the elements identification I consider both the nature of personal tasks and the users’ needs, which differ from the needs of organizations for which the BPMN was designed. In fact, an investigation conducted in [22] revealed that the users don’t want to deal with complex decision points, involving definition of conditional expressions, complex event management, or exceedingly complex process structures. The output of this phase is a set of language elements with their full description in terms of relevance and relationships (Table 5.1);

2. Generation of the language variants.

   In this phase, I take in input the elements defined in the previous phase and we produce a set of alternative syntaxes, the language variants. Language variants are generated from a sub set of the language elements.
To measure the effectiveness of the designed syntax we need it to be tested by intended users. The challenge is how to submit the elements to the users so that they can evaluate them in the best way. A unique syntax with all elements, for all users, is not possible. In fact, some elements (like alternative elements) cannot be used at the same time. Moreover, considering that we are modeling the experiment for end users, all elements maybe too confusing. A possible solution is to fix a number of elements for each variant and then take all possible combinations, but this creates a problem of finding people available to test a such high number of alternatives.

3. **Modeling of use cases by end-users.**

In this phase, the users test the language by modeling the assigned use case (a pair of a personal process with the language variant to be used to model it). During this phase, a logging system logs the users’ actions (such as the creation, modification, and deletion of an element) useful to derive quantitative data allowing to compare and evaluate the language variants. I also monitor the users by taking the notes on how they model the assigned processes, by direct observation. Those qualitative information complete the data gathered through a logging system for our analysis.

4. **User Questionnaire.**

For users profiling and segmentation, the users are asked to answer a questionnaire about their personal information: knowledge in the computer science field and about the ease of understanding the scenarios and modeling them with the assigned syntaxes. The questionnaire contains a free form field allowing the users to report their general feedback like the lack of elements or functionality in the syntaxes.

5. **Language evaluation.**

In this phase, I analyze the data collected in previous phases (the execution data stored in the logs and the notes taken during the use cases modeling phase, and the users’ evaluations given in the Questionnaire) to make an informed decision of the best variant based on those quantitative and qualitative data.

### 5.3 Selection of Language Elements

This section describes the selection of the elements which compose a notation for personal process modeling. The elements of this notation are a small subset of BPMN allowing to model the personal processes.

Considering the characteristics of such processes and the needs of end-users, we came up with the following elements: Task, Sequence, Parallel Execution, Conditional Execution, Events, Loop Execution, and Parameters. To understand how those elements can be used together to model a personal process, we have characterized each elements by specifying:

- The relevance (expressed as mandatory or not). The mandatory elements must be included in all the language variants otherwise the variant is invalid.

- The alternatives elements. For each element we enumerate the different options of a such element. For example the element *Parameter* can be a *global parameter*, a *one local parameter* or a *multiple local parameter*.
5.3. Selection of Language Elements

- The dependencies. For each element we identify the elements on which it depends on, if any. For example, when modeling a personal process the conditional execution uses the value of the parameter. Thus, conditional execution depends on parameter.

Table 5.1: Characterization of the elements of personal process modeling notation

<table>
<thead>
<tr>
<th>Element</th>
<th>Mandatory</th>
<th>Alternative</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Event</td>
<td>Yes</td>
<td>Start</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>No</td>
<td>Wait For</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wait Till</td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>Yes</td>
<td>N/A</td>
<td>Source and Target</td>
</tr>
<tr>
<td>Parallel Execution</td>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Conditional Execution</td>
<td>No</td>
<td>N/A</td>
<td>Parameters</td>
</tr>
<tr>
<td>Parameter</td>
<td>No</td>
<td>Global</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>One Local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple Local</td>
<td></td>
</tr>
<tr>
<td>Loop Execution</td>
<td>No</td>
<td>N/A</td>
<td>Conditional Execution</td>
</tr>
</tbody>
</table>

5.3.1 Task

Is an atomic activity included within a Process flow. A task is used when the work in the process cannot be broken down to a finer level of detail and it must be assigned to one or more actors.

5.3.2 Sequence

Sequence flow of execution is represented by arrows. An arrow connecting two tasks means that the task at the start of the arrow must be completed before starting the execution of the task at the end.

5.3.3 Parallel Execution

Parallel Gateways create parallel flows for a simultaneous execution of tasks.

5.3.4 Parameters

The parameters are used to store values submitted by actors. These values can be used in conditional gateways, printed in tasks bodies or just stored. We have identified three possible ways to use parameters:

- Global Parameters, available in all the processes directly or indirectly connected after the creating task;
- One-Local Parameters, available only in the tasks to which they have been propagated; and
Chapter 5. Language Simplification

- Multiple-Local Parameters. Similarly to one-local parameters, the multiple-local parameters are only available in the tasks to which they have been propagated. But, in this case, more than one parameter can be propagated.

5.3.5 Conditional Execution

The only way to make conditional execution is to create conditional parallel flows using conditional gateway. This notation allows using only the simple conditions. Therefore, the arithmetic operations and comparisons between parameters are not supported. The conditional gateway uses parameters to evaluate the condition.

5.3.6 Loop

The loop is a specific tool to create backward flows. It requires the conditional gateway to examine the loop conditions. The expressed conditions are mutually exclusive.

5.3.7 Events

An event is something that happens during the course of a process. The events are grouped in the following three categories:

- Start Event. It indicates where a process will start;
- Intermediate Event. An intermediate event indicates where something happens, an event, somewhere between the start and end of a process. In BPMN, those events are used to model events such as message-based communication among actors, flow control through exception handling, and delays expected within the business process. While those events are important to model business processes, some of them (e.g message-based communication) are not relevant in personal processes.

  The timer is the only relevant intermediate event in personal process. In fact, it could be useful to impose time constraints on the execution flow. We have introduced two time events: Wait For and Wait Till which express, respectively, the waiting for a generic time and the waiting of a specific date or time; and

- End Event. It indicates where a process will end.

The start and end event are considered mandatory elements, because processes need to have a start and end point. Conversely, intermediate events (Wait For and Wait Till) are not mandatory (Table 5.1).

5.4 Definition of Language Variants

To measure the effectiveness of the notation we need it to be tested by end-users through the modeling sessions. The challenge is how to submit the elements of the language to the users so that they can evaluate them in an efficient way. In our approach, this is done through a set of language variants: alternative syntaxes generated considering only a subset of the language elements. In the next paragraphs, we describe how we generate those variants.

We use the language variants, which use a reduced set of language elements, because it is easier (for end-users) learn and use a language composed by few elements rather
### 5.4. Definition of Language Variants

#### Table 5.2: The language variants

<table>
<thead>
<tr>
<th>Element</th>
<th>Alternative</th>
<th>Variant 1</th>
<th>Variant 2</th>
<th>Variant 3</th>
<th>Variant 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Start</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>End</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Task</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Parameter</td>
<td>Global</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One Local</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple Local</td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>Intermediate Event</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Parallel Execution</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Conditional Execution</td>
<td></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

than a complex one. Furthermore, it not always possible to test all the elements of a language together. For example, the alternative elements cannot be included in the same syntax at the same time.

However, testing all valid combinations of the elements becomes unfeasible due to the high number of the variants and consequently, the required high number of users to test them. Therefore, the dimension of the language variants (the number of the elements to include in each variant and the number of variants) is a compromise between the possible combinations of the elements and the number of users available to test them.

In our case, a good compromise between the number of elements and the number of the users available for the test was the following four variants (Table 5.2):

- Variant One: it makes use of sequence and parallel flows, events, global parameters, but no loops and no conditional flows.
- Variant Two: it makes use of sequence, parallel and conditional flows, global parameters, loops but no events.
- Variant Three: it makes use of sequence, parallel and conditional flows, events, one local parameter, but no loops.
- Variant Four: it makes use of sequence and conditional flows, events, multiple local parameters, but no loops and no parallel flows.

All the four language variants have the basic elements in common: the start and the end events, the task and the sequence routing. Those are mandatory elements for the personal process modeling language. We built a linear variant with global parameters, events, parallel routing but not conditional and loop gateways. We also built a more complex variant with multiple local parameters, conditional routing but no parallel gateway. The other two variants were instead more balanced.

---

1Indeed, we face similar trade-offs as those heavily studied in software testing. As such, a good strategy when defining the variants should be to maximize the coverage both in terms of combination of elements to tests and in terms of coverage of relevant usage scenarios for the language.
Chapter 5. Language Simplification

5.5 Modeling of Use Cases by End-Users

In previous section, we have defined the elements of a notation for personal process modeling and we have identified its four variants. The next step of our approach is to submit these variants to the end-users and conduct the experiment in which users model the assigned use cases. A use case represents a couple of a personal process with the language variant to be used to model it.

5.5.1 Scenarios

We have designed three scenarios based on the real life situations. Every scenario describes a process with the suggested activities and the general constraints.

- **Scenario A - Holiday with friends** You are organizing holiday with friends. You need to decide the transport, choose the hotel and plan possible tours. Then you have to draw up the budget and if it is included in the planned costs, you can proceed with reservations. The week before the departure, you will meet together to define the last details.

- **Scenario B - Association party** You are organizing a free-entrance party of your association. You need to publicize the event, and ask the authorization to the municipality if you registered at least 50 reservations. Then you can contact the catering service and engage the band. The event will start with the dinner at 8:00 pm followed by the concert at 9:30 pm if the dinner has finished.

- **Scenario C - Warehouse management** Your goal is to optimize the buying and selling of the raw materials, maintaining always 1000 units constant. Every week, the quantity of material in warehouse must be checked and then, according to the amount, you need to buy new material or sell it. The order must be approved by the accounting department. Once arrived, new stock must be placed and cataloged.

In addition, we have developed a test scenario, used at the beginning of the modeling sessions to explain to the users how the tool works and to let them get confidence with it.

**Scenario Test - Basket tournament** You are organizing a basket tournament. You need to collect the registrations, schedule the matches, rent the playgrounds and find the sponsors.

5.5.2 Use Cases Definition

A use case represents a pair of a scenario with a language variant to be used to model it. The use cases are based on the relevant scenarios of the domain. Having 4 language variants and 3 scenarios, overall there are 12 possible pairs, use cases (Table 5.3).

5.5.3 Assigning Use Cases to Users

We have decided to submit two use cases to each user. The submission order of the two use cases is important because in the modeling of the second use case, the user will apply the knowledge acquired in the first one. In order to overcome unintended effects and to have balanced experiments, all the possible use cases pairs have to be studied considering both the order and the scenario-variant coupling. For that purpose, we use
5.5. Modeling of Use Cases by End-Users

Table 5.3: Use cases

<table>
<thead>
<tr>
<th>Use case</th>
<th>Language variant</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>C</td>
</tr>
</tbody>
</table>

the Graeco-Latin square theory [66]. Applying the Graeco-Latin square theory to the use cases, we obtained the result shown in Figure 5.2.

Each cell of the table, depicted in Figure 5.2, represents a single experiment made by two use cases to be assigned to a single user. In each row of the table, every variant and every scenario appears the same number of times, and every use case appears one time in the first position and one time in second position. The invalid experiments are highlighted in red. We consider invalid an experiment which has the same scenario and/or the same variant in both tests. For example, the experiment [2,3] is invalid because of the repetition of scenario A, while the experiment [5,9] is invalid because of the repetition of the first variant. Excluding the red cells, the remaining green cells can be grouped as shown in Figure 5.3.

The second and the fifth rows have six cells each that are mirrored repetitions of the other six cells ([3,5] and [5,3], [4,6] and [6,4], etc). Let’s call the remaining rows A, A’, B, B’, C and C’ (Figure 5.4). We can see that A’ has the same couplings of A but with use cases in the inverse order. The same holds for B’-B and for C’-C.

5.5.4 Modeling Sessions

Each modeling session of the use cases by end-users has been conducted following this exact procedure in five steps:

1. **Introduction**
   First, we have introduced the modeling tool to the users and we have presented them its purpose. Then we have explained them the objective of the experiment: to test the effectiveness and ease of use of the adopted language variant to model the small processes, in daily life or in small and medium-sized enterprises.

2. **Registration**
   The users logged in to the tool using one of their social accounts.

3. **Instruction**
Chapter 5. Language Simplification

4. Experiment
   Each user had to model two use cases. In this step the user could read the manual of the first assigned variant followed by the description of the coupled scenario. At this point he could start modeling the process. Once the first process was modeled, the user could proceed with the second use case in the same way described before.

5. Closure
   At the end of the test, after the user had submitted both models, he was asked to answer a questionnaire about his personal information, his knowledge in the computer science field, the ease of understanding the scenarios and modeling them with the assigned language variant. He was also asked to report the lack of elements or functionality in the assigned language variant.

Figure 5.2: Graeco-Latin square of use case pairs

The users read the main manual of the editor which describes how to make the basic actions of creation, modification and deletion of elements. Then, a short test of 3 minutes took place under our supervision in order to take confidence with the basic functionality of the tool. This was done using a test scenario, equal for all the users.
5.5. Modeling of Use Cases by End-Users

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.7</td>
<td>11.8</td>
<td>4.5</td>
<td>1.10</td>
<td>2.11</td>
<td>3.12</td>
</tr>
<tr>
<td>2</td>
<td>3.5</td>
<td>4.6</td>
<td>1.11</td>
<td>2.12</td>
<td>7.9</td>
<td>8.10</td>
</tr>
<tr>
<td>3</td>
<td>4.7</td>
<td>1.12</td>
<td>6.9</td>
<td>7.10</td>
<td>8.11</td>
<td>5.4</td>
</tr>
<tr>
<td>4</td>
<td>6.11</td>
<td>7.12</td>
<td>8.1</td>
<td>9.2</td>
<td>10.3</td>
<td>11.4</td>
</tr>
<tr>
<td>5</td>
<td>7.1</td>
<td>8.2</td>
<td>9.3</td>
<td>10.4</td>
<td>11.5</td>
<td>12.6</td>
</tr>
<tr>
<td>6</td>
<td>8.3</td>
<td>9.4</td>
<td>10.5</td>
<td>11.6</td>
<td>12.7</td>
<td>1.8</td>
</tr>
</tbody>
</table>

**Figure 5.3:** Valid experiments

<table>
<thead>
<tr>
<th>A</th>
<th>10.7</th>
<th>11.8</th>
<th>4.5</th>
<th>1.10</th>
<th>2.11</th>
<th>3.12</th>
<th>8.9</th>
<th>5.2</th>
<th>6.3</th>
<th>7.4</th>
<th>12.1</th>
<th>9.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A'</td>
<td>4.7</td>
<td>1.12</td>
<td>6.9</td>
<td>7.10</td>
<td>8.11</td>
<td>5.4</td>
<td>10.1</td>
<td>11.2</td>
<td>12.3</td>
<td>9.8</td>
<td>2.5</td>
<td>3.6</td>
</tr>
<tr>
<td>B</td>
<td>6.11</td>
<td>7.12</td>
<td>8.1</td>
<td>9.2</td>
<td>10.3</td>
<td>11.4</td>
<td>12.5</td>
<td>1.6</td>
<td>2.7</td>
<td>3.8</td>
<td>4.9</td>
<td>5.10</td>
</tr>
<tr>
<td>B'</td>
<td>8.3</td>
<td>9.4</td>
<td>10.5</td>
<td>11.6</td>
<td>12.7</td>
<td>1.8</td>
<td>2.9</td>
<td>3.10</td>
<td>4.11</td>
<td>5.12</td>
<td>6.1</td>
<td>7.2</td>
</tr>
<tr>
<td>C</td>
<td>1.7</td>
<td>2.12</td>
<td>3.5</td>
<td>4.10</td>
<td>5.11</td>
<td>6.4</td>
<td>7.9</td>
<td>8.2</td>
<td>9.3</td>
<td>10.8</td>
<td>11.1</td>
<td>12.6</td>
</tr>
<tr>
<td>C'</td>
<td>1.11</td>
<td>2.8</td>
<td>3.9</td>
<td>4.6</td>
<td>5.3</td>
<td>6.12</td>
<td>7.1</td>
<td>8.10</td>
<td>9.7</td>
<td>10.4</td>
<td>11.5</td>
<td>12.2</td>
</tr>
</tbody>
</table>

**Figure 5.4:** Valid experiments without duplicates

### 5.5.5 Monitoring

During the experiments, we monitored users’ actions in order to collect the data indispensable to assess language variants. The monitoring was done through: direct observation, noting every user’s activity and comments made while modeling the assigned processes; and through a logging system. The logging system was limited to the graphical editor and each log row contains a code name for the tracked action, a field used for additional info, and the target of the action. The logs have been stored in a structured database from which the desired data are extracted by means of SQL queries.

We have prepared a set of basic queries that retrieve the: (i) time required to create each process; (ii) number of times an element has been edited; (iii) number of elements created; (iv) number of times an element has been moved; (v) number of delete; (vi) number of process savings; (vii) number of validation requests; and (viii) number of invalid connections. Although the tool checked for correctness of the model and did not allow to create wrong models, it recorded the number of invalid connections as the
number of times the users attempted at drawing wrong connections between elements. Starting from these basic queries, we have built the composite queries extracting derived data used to compare and to evaluate the language variants. All these data, together with the opinions and feedback collected from the questionnaire, must be analyzed to understand each variant pros and cons and determine a good compromise.

5.5.6 Execution Data Analysis

The experiment involved 24 users. 21 are men, while only 3 are women. 19 users are in the 18-30 years range, 2 users in the 31-50 range and 3 users in the 51-70 range. Half of the users are students, 7 have high skills in computer science while 9 users define themselves as basic PC users. Overall, the users are not experts in BPMN. In fact, only 2 users claim to use it regularly, 8 know it but they do not use it while the remaining 14 have never heard about it.

5.5.7 Durations

From the analysis of the times of the single experiments (Figure 5.5) we notice that processes modeled with the first variant are those done faster, with an average duration of 16’27”. On the contrary the fourth variant is the slower, with an average time of 22’02”, about 34% slower than the first (5’35” more). The second and the third variants are instead quite similar between them, about 16% and 15% slower than the first. Looking at the standard deviations, the fourth and first variants have higher values. All the delivered processes are correctly validated except one made with the second variant. It is interesting to notice that 19 times over 24, the second test has been modeled in less time than the first one, with an average of 7’30” less. This is probably due to the fact that during the first test users need more time to take confidence with the editor. In the remaining 5 times, users took an average of 3’20” more. Also the variants play a role in the difference of time between the two tests. In fact, users who play first with the fourth variant, and then with the first variant, take on average 12’43” less, while users who first used the first variant and then used the second variant, take on average 1’27” less. It seems that the fourth variant is more powerful then the first but is also heavier to use, so it takes more time. The first variant is simpler, offers minor possibilities and is faster to use.

Number of Elements Creations and Deletions

The processes modeled with Variant One have few elements, thus they are smaller and simpler. Instead, Variant Four has the most number of creations and is the richer one. Variant Two and Variant Four seem to have same number of elements used (Figure 5.6a). The number of deletions is quite constant for all the variants, a little less in the first and a little more in the second.

The most used elements are of course tasks and connections. Those most used elements are omitted in (Figure 5.6b) to better highlight other elements variations. The Wait For events are rarely used and have a high percentage of deletions in all the variants in which they appear, respectively 60%, 80% and 67%, while the Wait Till events are preferred and more used. The Parallel Gateway is less used than the Conditional Gateway. In the first variant there is only the Parallel Gateway, in the fourth there is
5.5. Modeling of Use Cases by End-Users

![Diagram of Experiments duration per single user](image)

**Figure 5.5:** Experiments duration per single user

![Diagram of Language variant quantity](image)

**Figure 5.6:** (a) Shows the average number of creations and deletions per process while (b) shows the average number of elements per process

only the *Conditional Gateway*, while in the other two they appear both. In the second variant the *Parallel Gateway* is used twice as often as the Conditional. In relation to these data we also need to point out that variants three and four are those with a greater number of created elements overall. The *Loop*, with an average use of 1.33 per process, is the most used element in the processes where it is available, namely those modeled with variant two.

**Validation Requests**

During all the tests, users have used the validation button to check the processes correctness. Generally, the processes were correct but sometimes errors were found and users had to validate again the process. Globally, all the variants have almost the same number of validations. The highest percentage of validations with errors is the one related to the Variant One, immediately followed by Variant Three. Variants One and Variant Four are those with the lowest percentages of false validations.
Chapter 5. Language Simplification

Figure 5.7: Variants difficulty

Editing Elements

The average time per process spent by users editing elements is approximately the same for all the variants except for the Variant One that is significantly smaller. On average, the time spent modifying single elements is more in the Variant One, though with higher standard deviation, while less time was spent on Variant Four. On the other hand, the number of element changes is greater in Variant Four, while it is smaller for the Variant One. The high number of changes for variants with local parameters characterized however by short editing times. These characteristics should be caused by the need to manually propagate local parameters. The data related to the events showed that they were edited practically in the same way. Conditions required on average, more time to be configured with Variant Four.

5.6 Questionnaire

This section resumes the answers gathered from the 24 users involved in the experiments. The users’ feedback complement quantitative measures presented in Section 5.5.6 Execution Data Analysis.

5.6.1 Scenarios difficulty

Each scenario was tested sixteen times. Both scenarios A and B were judged mainly easy with 13 and 12 votes each and the remaining votes on medium level. Scenario C resulted to be the hardest with only 8 votes on easy level, 7 on medium level and 1 on hard level.

5.6.2 Language variants difficulty

Each variant was tested twelve times. Variants One, Two and Three were equally judged with 7 votes on easy, 4 on medium and 1 on hard difficulty. Instead, Variant Four is
5.7. Language Evaluation

perceived harder since users gave only 2 votes on easy, 8 on medium and 2 on hard (Figure 5.7).

5.6.3 Cases difficulty

The easiest combination of variants and scenarios resulted to be Variant Three and Variant Four with scenario A, and Variant One with scenario B. The worst was Variant Three with scenario C.

5.6.4 Variants deficiencies

The most noticed deficiency was the lack of the Loop element. In fact, in 15 tests, users claimed the necessity to have the loop to correctly model the process they had in mind. Particularly, its absence was felt more in the third variant.

The Conditional Execution is included in all the variants except for the first, and right in this, in 8 tests over 12, users wanted it. The Parallel Execution does not exist only in the fourth variant end its absence was notice only in 3 tests over 12 tests. The lack of the events has not gone unnoticed in Variant Two.

Users noticed also deficiencies on parameters types: someone claimed Boolean parameters but then he resolved using the textual type; other users would have preferred to create lists of parameters and treat them as arrays. In some cases it would have been useful to compare two parameters inside a condition, or sum them or compare them with a value. Using the third variant, in two cases users would have needed to propagate more than one local parameter, while in other two cases they would have preferred to have global parameters that are more comfortable to use. In the fourth variant instead, in three tests the propagation of multiple local parameters was found too heavy.

After all, even if every variant has its pro and cons, all the users have succeeded to model the assigned processes, sometimes using alternative, weird but functional methods to bypass the restrictions.

5.7 Language Evaluation

By analyzing the data related to the different aspects of the modeled processes during the experiments (the users evaluations given in the questionnaire, the notes taken during the experiments, and the execution data), we can state that there is no language variant which is clearly better than the others in absolute terms. This is quite common in any realistic setting, and it’s the reason why the problem of selecting the best language is not a trivial task at all (notably, it cannot be reduced to simply choosing the variant with the most used concepts), also because different relevance or priority may need to be put on one aspect or the other when finalizing the choice for the given vertical scenario.

However, the information collected and combined together through the various means proposed in our approach, provide evidence and quantitative data allowing the decision to be made more objectively and in an informed way.

The main methodological guideline consists in defining two dimensions in the decision making process: (i) Language evaluation. Besides evaluating independently each single language element, it’s also important to consider how they actually integrate with each other in a comprehensive tool, i.e., the language variant. (ii) Element evaluation.
Chapter 5. Language Simplification

<table>
<thead>
<tr>
<th>Language</th>
<th>BPMN</th>
<th>Variant One</th>
<th>Variant Two</th>
<th>Variant Three</th>
<th>Variant Four</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of element</td>
<td>52</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5.4: Comparing the elements number of the initial language, BPMN, with its simplified version, the language variants. BPMN provides 52 modeling elements while the four variants use 7, 8, 8, 7 elements respectively.

This dimensions suggests to evaluate each element in the language independently, so as to compare the different available options for its implementation and get oriented towards the potentially best selection of elements.

There is no prescribed order or ranking of importance between these two aspects, as the final decision may very well require some looping over the two, until a satisfactory selection is reached for both. We report here the evaluations and analysis done in our case study, so as to provide hints on the possible aspects to be taken into account in the decision.

5.7.1 Language Variants Evaluation

Each language variant exhibits a set of pros and cons, which again can be distilled by analyzing both the quantitative measures taken during the experiments, and the feedback of the users through the questionnaire. Probably the best variant would be the one achievable choosing the best options of each element based on the previous analysis. However, overall language characteristics must be considered too.

All the four language variants designed in this work are simpler, in terms of the number of used elements, with respect to BPMN. In fact, Variant One and Variant Four use only 13.5% of BPMN elements while Variant Two and Variant Three use 15.4% of BPMN elements. In our specific case, the number of used elements is not a strong discriminating factor since all variants use more or less the same number of elements (Table 5.4). Here follow the resuming considerations about the variants, along with a pros and cons list of each one.

5.7.2 Variant One

It is the quickest and leanest in the processes creation, but in many cases it has turned out to be too poor and ineffective. It is more suitable to describe very easy and linear processes and it is too limited if there is the need to increase the modeling detail. Its pros and cons are reported in Table 5.5.

5.7.3 Variant Two

It is the only variant which presents all the gateway types: particularly the Loop Gateway has proved to be really useful in the modeling of the more complex passages of the processes. Its pros and cons are reported in Table 5.6.

5.7.4 Variant Three

This variant represents a good compromise between ease of use and descriptive power but the simplicity of a single parameter is, in practice, a great limit. Its pros and cons are
5.7. Language Evaluation

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| • Small number of elements to learn and a few rules to use them.  
• Processes are modeled faster with a smaller number of elements and deletions.  
• Shorter process design time.  
• No connection errors were done during the experiments.  
• A few validation errors were done during the experiments.  
• Smaller number of validation requests and lower percentage of wrong models submitted for validation. | • Lack of specific tools to express the conditional execution.  
• Harder in medium complex and complex processes modeling.  
• Harder in medium complex and complex processes modeling.  
• Poor expressive power. |

<table>
<thead>
<tr>
<th>Table 5.5: Language Variants Evaluation: Pros and Cons of Variant One</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>
| • The loop gateway makes possible to model iterative execution.  
• Average modeling time.  
• Number of used elements below the average.  
• Small number of connection errors. | • Lack of specific tools to express time events.  
• Higher percentage of validation errors.  
• Higher number of validation requests. |

<table>
<thead>
<tr>
<th>Table 5.6: Language Variants Evaluation: Pros and Cons of Variant Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>
| • Lower Wait Till and Conditions editing times.  
• Small number of Wait Till and Conditions changes.  
• Lower number of users suggestions in the Questionnaire. | • Number of connection errors over the average.  
• Each time a parameter is added it must be manually propagated, if needed. |

<table>
<thead>
<tr>
<th>Table 5.7: Language Variants Evaluation: Pros and Cons of Variant Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
</tr>
<tr>
<td>------</td>
</tr>
</tbody>
</table>
| • Lower Wait Till and Conditions editing times.  
• Small number of Wait Till and Conditions changes.  
• Lower number of users suggestions in the Questionnaire. | • Number of connection errors over the average.  
• Each time a parameter is added it must be manually propagated, if needed. |

5.7.5 Variant Four

This variant turned out to be the worse one, and was the one more criticized by users. Its pros and cons are reported in Table 5.8.

5.7.6 Elements Evaluation

Single element evaluation should take into account quantitative performance of the element in the experiment, as well as hints collected explicitly from users. Indeed, data collected through the logging of the experimental modeling phase may have little meaning if not complemented by the users opinions and descriptions of the problems. Therefore, also the users evaluations given in the questionnaire and the notes taken during the experiments must be taken in consideration. Keeping in mind all these aspects, we can give a resuming evaluation of the single elements of the four variants in our scenario.
Chapter 5. Language Simplification

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Small number of validation errors.</td>
<td>• Connection errors above the average.</td>
</tr>
<tr>
<td>• Lower number of validation request with lower percentage of invalidity.</td>
<td>• Each time a parameter is added it must be manually propagated, if needed.</td>
</tr>
<tr>
<td></td>
<td>• Bigger number of elements used.</td>
</tr>
<tr>
<td></td>
<td>• Longer processes creation times.</td>
</tr>
<tr>
<td></td>
<td>• Longer elements editing times.</td>
</tr>
<tr>
<td></td>
<td>• Higher number of editing on elements.</td>
</tr>
<tr>
<td></td>
<td>• The hardest to use according to Questionnaire results.</td>
</tr>
<tr>
<td></td>
<td>• Greater number of suggestions in the Questionnaire.</td>
</tr>
</tbody>
</table>

| Table 5.8: Language Variants Evaluation: Pros and Cons of Variant Four |

Wait For Event

It has been by far the least used element and the most deleted. Sometimes, instead of using it, users preferred to use the time constraints inside the tasks, misinterpreting their meaning.

Wait Till Event

Compared to the Wait For event, the Wait Till event has been used and appreciated by the users, and it has been useful to resolve the proposed scenarios.

Parallel Gateway

It is a basic element that has been frequently used when it was available and missed in the variant where it was not. Sometimes it has been replaced by the Conditional Gateway using equal conditions on the branches.

Conditional Gateway

It has turned out to be fundamental for its great descriptive power. Users have felt the lack of it when it was not available.

Loop Gateway

Looking at the collected data, the Loop has been the most relatively used element, proving to be very useful to resolve the critical aspects of the modeling. When available, it has been really appreciated, while, when missing, users claimed its need and have found harder the process modeling. Conversely, the Loop element has introduced some more errors, so it requires a little more attention in its use.

Global Parameters

Overall, global parameters were easy to understand and much appreciated, despite an initial difficulty of some users that were not confident with the concept of variable and parameter in computer environment (this consideration applies also to local parameters).
5.8. Generalization

One Local Parameter
It has shown its limit when there was the need to receive more than a parameter in a task. Users found not intuitive to propagate and receive the parameter.

Multiple Local Parameters
They have been judged more useful than the one local parameter because they increased the modeling efficiency, but on the other hand, they were criticized because of the heaviness they introduce in the notation.

5.7.7 Language Variant Selection
The selection of the best language variant depends highly on the domain needs. However, the winning variant should be the one having the following properties:

- Simplicity. The language variant should be simple to understand and easy to use. We assess this property through the analysis of the quantitative data (such as required time to create the process, number of connection errors and validation requests) gathered during modeling sessions and qualitative information collected through user questionnaire and notes.

- Completeness with respect to domain requirements. The winning variant should allow the modeling of all relevant aspects of the domain. We assess this property through analysis pros and cons of each variant and the evaluation of single elements.

The language Variant One fits the most the above-mentioned properties.

Simplicity. The analysis of pros and cons showed that processes modeled with Variant One were done faster, using small number of elements and they do not presented any connection error. Furthermore, the validation ratio (number of validation errors/number of validation requests) was low.

Completeness. Single elements evaluation showed that all the elements of Variant One are well understood and users required them when missing. The pros and cons analysis showed that variant One has poor expressive power. However, this is not a problem in personal process. As said so far, the selection of the best variant depends on the domain. In next paragraph, we illustrate how the winning variant could have been another one given the same settings but changing only the domain.

5.8 Generalization

In this section, and based on the experience reported so far, we provide some hints on how the described simplification process can be generalized to be useful in the simplification/personalization of other modeling languages.
Chapter 5. Language Simplification

The generalization involves mainly the initial parts of the process where we need to reduce the scope of the problem by first choosing a base language to simplify and the set of relevant elements to evaluate. Once this is done, the rest of process follows the same schema of Figure 5.1.

While in the case study described in this paper, the choice of BPMN as the starting point was an obvious choice as the standard and widely adopted language in the process modeling area, but things may not the same in other domains. If there are several candidate languages (to be found, for instance, by looking at systematic reviews available for the domain), the selection of the base language to simplify must be done objectively based on some predefined criteria, the selection dimensions. While these dimensions may be decided by each company, clearly, aspects like the popularity of the language, its tool support or the availability of predefined extension mechanisms should be taken into account since they will for sure facilitate the simplification process.

Then, you should proceed with the identification of the language elements to evaluate in the language variants step. In general, the key to do well this phase is to have a deep understanding of the needs of the end-users: what are their requirements? what language elements are more likely to fit those requirements? Those elements would constitute the starting set of candidate elements to focus on. Next we should explore how they are related in the original language metamodel among themselves and with other (auxiliary) elements to enrich that set. Once this identification phase is completed we can proceed with the definition of the variants.

Once this is done, the rest of the process described in the paper can be easily adapted to any language. For instance, to define the language variants, the only language-specific part consists in the identification of the dependencies between the language elements to evaluate (captured by looking at the metamodel definition as indicated in the previous step). Similarly, the process to design the use cases should follow the same recommendations as for the BPMN scenario (mainly to make sure that the use cases cover the variety of situations we want to evaluate). The data analysis part is not language-dependent so its application is straightforward.

5.9 Conclusion

In this chapter, we reported on a field study aimed at the simplification of a business process model language for making it suitable to end-users. Our simplification process relies on: (i) the selection of the language elements to simplify, (ii) generation of a set of language variants for those elements, (iii) measurement of effectiveness of the variants through modeling sessions performed by end-users, and (iv) extraction of quantitative and qualitative data for guiding the selection of the best language refinement. We described the experimental setting, the output of the various phases of the analysis, and the results we obtained from users.

The results of our study, showed that the followed approach provides evidence and qualitative and quantitative data enabling the selection of the best language variant to be taken more objectively and in an informed way.
CHAPTER 6

User Interaction Modeling

Front-end design of mobile and IoT-based applications is a complex and multidisciplinary task, where many perspectives intersect and the user experience must be perfectly tailored to the application objectives. However, development of the user interactions is still largely a manual task, which yields to high risks of errors, inconsistencies and inefficiencies. The availability of a platform-independent user interaction modeling language can bring several benefits to the development process of mobile and IoT-based application front-ends, as (i) it improves the development process, by fostering the separation of concerns in the user interaction design, thus granting the maximum efficiency to all the different developer roles; (ii) it enables the communication of interface and interaction design to non-technical stakeholders, permitting early validation of requirements. This chapter presents the modeling language defined for modeling the user interaction of mobile applications and its IoT-extension, allowing to express the user interaction with IoT systems.

The rest of the chapter is organized as follows: Section 6.1 presents a mobile modeling language named Mobile IFML [27]. Mobile IFML is a platform independent modeling language designed for expressing the content, user interaction, and control behaviour of the front-end of mobile applications. It is an extension of a OMG standard called Interaction Flow Modeling Language (IFML) [24], designed to address the specific requirements of mobile devices. Section 6.2 presents the new components defined to model the interactions between the terminal (the device used to access the IoT system) and the IoT devices. Section 6.3 concludes the chapter presenting our design methodology—the modeling tool and a set of design patterns allowing the modeling of recurrent problems in user interaction modeling for both mobile and IoT-based applications.
Chapter 6. User Interaction Modeling

Figure 6.1: IFML mobile extension: Screen, ToolBar and mobileComponent.

6.1 Mobile IFML

Mobile applications have rich interfaces that resemble on a smaller scale those of full-fledged desktop applications, possibly with additional complexity of the interaction patterns, at the purpose of exploiting at best the limited space available. This aspect, together with mobility and availability of sensors, such as camera and GPS, introduce features that are best captured by providing mobile-specific extensions of a platform-independent modeling language like IFML.

This section presents a platform independent modeling language defined to model the front-end of mobile applications. We call this modeling language Mobile IFML. Mobile IFML has been defined as a mobile-specific extension of a OMG Standard for the user interaction modeling, the Interaction Flows Modeling Language (IFML). The proposed extensions come from extensive modeling experience on mobile applications using the IFML standard, which covered both modeling of several existing mobile applications (including CamScanner, Instagram, iPhone Gallery, Twitter, RedLaser and many others) and design and implementation of new mobile applications for industrial customers. Those extensions address both the components of the user interface and the events.

6.1.1 Mobile Containers and Components

This section describes the concepts added to IFML in order to model the components that characterized the mobile context (as shown in Figure 6.1).

Screen

A new class called Screen has been defined to represent the screen of a mobile application. Since the screen is the main container of a mobile application, it extends the core class ViewContainer of the IFML standard.

ToolBar

The class ToolBar represents a particular subcontainer of the screen. It may contain other containers and may have on its boundary a list of events. It extends the core class ViewContainer of IFML standard.
6.1. Mobile IFML

Mobile System

A characteristic trait of mobile interfaces is the utilization of predefined ViewContainers devoted to specific functionalities (including Notifications area and Settings panel). These system level containers provide economy of space and enforce a consistent usage of common features. The MobileSystem stereotype has been defined to distinguish these special ViewContainers. A ViewContainer stereotyped as MobileSystem denotes a fixed region of the interface, managed by mobile operating system or by another interface framework in a cross-application way.

Mobile Component

The class MobileComponent denotes the particular mobile view component such as buttons, images, icons etc. A MobileComponent is subject to user events described in the next paragraphs. The MobileSystem stereotype can be applied also to ViewComponents to highlight that the interface uses the components built-in in the system (as shown in Figure 6.6 usage of photo viewer).

6.1.2 Mobile Context

The context assumes a particular relevance in mobile applications, which must exploit all the available information to deliver the most efficient interface. Therefore, the context
Chapter 6. User Interaction Modeling

![Diagram of mobile system events]

Figure 6.4: IFML mobile extension: Mobile system events

must gather all the dimensions that characterize the user’s intent, the capacity of the access device and of the communication network, and the environment surrounding the user. A new class MobileContext extending the Context has been defined to express the mobile contextual features.

6.1.3 Mobile Events

In this section we describe the new event types that are defined within IFML for the mobile context. The conducted experiments identified three categories of mobile events:

1. Events generated by the interaction of the user such as tap and hold, swipe, etc.
2. Events triggered by the mobile device features such as sensors, battery, etc.
3. Events triggered by user actions related to the device components such as taking a photo, recording a video or using the microphone.

First, a new class MobileUserEvent allowing the modeling of the mobile user events have been defined. MobileUserEvent extends the core class ViewElementEvent of the IFML standard. The following classes extend MobileUserEvent for modeling the specific mobile user events: DragDrop; DoubleTap; Swipe; Pinch; Spread; Touch; LongPress; Scroll; and Shake. Each class represents an event related to the gesture which triggers it. The screens in Figure 6.2 show an example of the usage of the LongPress gesture allowing the user to manage the selected list. Figure 6.3 shows a fragment of IFML model for lists management. When a user performs the LongPress gesture on one element of the list a pop up containing information of the selected element is shown allowing her to edit or delete the list.

Mobile System Event

A new class MobileSystemEvent extending SystemEvent has been defined to express the mobile system events. The following classes extend MobileSystemEvent for specific system events:

- BatteryEvent, describing the events related to the state of the battery.
- StorageEvent, describing the events related to the archiving capacity.
6.1. Mobile IFML

Figure 6.5: IFML mobile extension: (a) MobileAction, the extension of the IFML Action to address specific mobile actions. (b) MobileActionEvent, the extension of the IFML ActionEvent class.

- **NotificationEvent**, grouping the events related to the generic notifications handled by the operating system.
- **ConnectionEvent**, representing the events related to the connection state of the device.
- **SensorEvent**, a class which defines the events related to the sensors of the device. The SensorEvent extends ExternalEvent IFML core class. The most commonly used sensors are proximity sensor, motion sensor, magnetometer sensor, Gyroscope and position sensor. The classes PositionEvent, MotionEvent, AccelerationEvent, ProximityEvent and RotationEvent extend the SensorEvent to represent the events related to the specific sensors.

**Mobile Action Event**

MobileActionEvent class has been defined to model the events triggered by a mobile action. Among mobile actions, we have actions related to the photo camera such as the **Shoot** action and actions related to microphone as reported in Figure 6.5. Figure 6.6 shows example of such events. A user takes a photo with the device’s photo camera and the application displays the product corresponding to the taken photo if any. Once the photo is available, a screen asking the user if he wants to use or retake the photo is displayed. The **photo available** event is associated to the action **shoot**.

6.1.4 Running Example

To demonstrate the applicability and the expressive power of the proposed extensions, this section exemplifies their use by modeling some of the functions of RedLaser\(^1\), a shopping app available for iPhone, Windows Phone and Android. Figure 6.7 shows the data model of the RedLaser application. In RedLaser the **products** are organized in **categories**. Each product has one or more **prices**. A user can create a **list** of the products he likes. The application allows the user to **review** a product. The RedLaser user interface consists of a top level container, which is logically divided into three alternative sub containers one containing the recent and popular products, the second

\(^1\)http://redlaser.com/
Chapter 6. User Interaction Modeling

Figure 6.6: IFML Mobile extension: Example of usage of MobileAction (Shoot), MobileActionEvent (Photo available) and MobileSystem stereotype.

Figure 6.7: Domain model of the RedLaser mobile app

one containing the history of the application and the last one allowing the user to save its favorites (as shown in Figure 6.8). The IFML model of the top screen comprises the ToolBar of the app and three screens in alternative: (i) Home, the default screen accessed when the user starts the app; (ii) History, that contains the log of the application; and (iii) Lists, a screen that allows the users to save their favorite products.

Product Searching

The app allows the user to search a product in four different ways:

1. Keyword search;
2. Photo based searching. The input of the searching system is a picture taken through the camera of the device;
3. Voice based searching; and

Figure 6.9 shows the screens allowing the searching of a product. Figure 6.10 shows a piece of IFML model for KeywordSearch and a voice based searching. The user can use the product selection event associated to each product on the lists of retrieved products to see its details as shown on the Figure 6.11.
Figure 6.8: Case study: (a) the home screen of RedLaser application; (b) IFML model showing logical organization of RedLaser application.

Figure 6.12 shows a piece of IFML model of the product overview. This model shows also the usage of the swipe user event to navigate among the sub screens of overview.
Chapter 6. User Interaction Modeling

Figure 6.9: Case study: UI for product searching. (a) Shows the user interface for keyword based search; (b) Shows the interface allowing the user to interact with the device’s microphone in voice based search.

Figure 6.10: Case study: IFML model corresponding to product searching shown in Figure 6.9. The model consists of input forms allowing the user to enter the keyword, Search Product actions and scrollable lists displaying the retrieved products.
6.1. Mobile IFML

Figure 6.11: (a) This screen shows the list of the prices available online for the selected product. It is the default screen of product overview; (b) The local is accessed from the online and related screens by using SwipeRight and SwipeLeft gestures respectively; (c) Shows the list of other product related to the selected one. (d) Displays the details of the selected product; (d) Contains a list of reviews related to the selected product.

Figure 6.12: Case study: IFML model corresponding to Online and Local tabs of product overview shown in figure 6.11. It consists of ProductDetail ViewComponent and two alternative lists of prices. The default one for online prices and the second displaying the local prices of the selected product.
Chapter 6. User Interaction Modeling

6.2 IoT Integration

User interaction plays a crucial role in every system. This is true also for IoT systems, although this aspect has been frequently neglected. The intelligent things connected together can cooperate and exchange information, but their ultimate goal is to provide value to people. Such value can be perceived only through appropriate user interfaces, which visualize information (through dashboard, reports, or infographics), let user navigate the information, and also interact with the devices, by setting properties or regulating their behavior.

As introduced in Chapter 2, the interactions between the user and the IoT systems can be logically divided in two phases:

1. User ⇌ Terminal communication. This phase represents the interactions between the user and the terminal used to access the IoT system.

2. Terminal ⇌ IoT devices communication. This phase represents the interactions between the terminal and the IoT devices.

The first phase of the user interaction with IoT systems can be modeled using the IFML standard and its current extensions, especially the Mobile IFML defined in Section 6.1 since mobile devices are among the most used terminals to access the IoT systems.

This section is dedicated to the second part of the user interaction with the IoT system: Terminal ⇌ IoT devices interactions. It presents the new components added to IFML to model both the events and actions associated to the IoT devices. The proposed extensions have been defined through the following phases:

1. Mapping of the IoT adoption and its current applications. This phase produced the main concepts and their relationships summarized in the IoT Reference Domain Model depicted in Figure 6.13.

2. Extraction of common use cases of the IoT. The use cases identified during this phase include: device management, device discovery (or search), interaction with devices, and information collection (highlighted in Figure 6.14). Those use cases come from an extensive investigation on IoT applications available on the market and what is expected to be the user interface of the IoT applications in different areas of their application. We present those use cases using the following schema: for each use case we provide a description, primary actor, and main tasks (as reported in Table 6.1). The user of the IoT-based applications could have different roles, defined as a set of allowed actions that the user can perform. The main roles are:

   • Administrator, the user who has the access right to the whole system, including the external systems;
   • Performer, the user who can manage and interact with the devices of the local network;
   • Viewer, the user who can display the information of the devices or the information about the environment motored by those devices.

3. Definition of a set of new IFML components allowing the modeling of the IoT user interactions, described in the sequel of this section.
6.2. IoT Integration

4. Definition of a set of reusable design patterns (presented in Section 6.3.3).

6.2.1 IoT Events

In this section we describe the new events defined as IFML extension for the IoT domain. Those events are grouped in: events from devices, and events associated to IoT actions.

Events from Device

The IoT devices emit specific signals containing information about their status or about what they are monitoring. Those signals are captured by specific catching events and sent to the users (terminal) in form of notifications. Those events are grouped into two categories:

1. Single Information Event. It is an event which captures every single message from the device it is listening to. A new class SingleInformationEvent extending SystemEvent of the IFML standard has been defined to model the events which capture every single message sent by a specific device. The usage of this event is described in the Figure 6.16.

2. Approaching Event. It is an event that allows the capturing of a first signal sent by each device it is listening to. The approaching event is used when the information transmitted by each device must be shown to the user only once: each time the device is detected for the first time by the terminal or when the device went out of the coverage range and comes back in. A new class, ApproachingEvent, extending SystemEvent has been defined to model the approaching events. The model depicted in Figure 6.17 exemplifies the usage of approaching event.
### Table 6.1: IoT use cases.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Description</th>
<th>Actor</th>
<th>Main tasks</th>
</tr>
</thead>
</table>
| Configure Access and     | Allows the application owner or the administrator, to set the access rights for users, teams or roles. | Administrator          | • Manage users, teams & roles  
  | Permission               |                                                                          |                        | • Access configuration  
  |                           |                                                                          |                        | • Permission configuration  
  |                           |                                                                          |                        | • Visualize information  |
| Interact with Devices     | Allows the user to send a set of operations to the devices, which are in charge to perform them. | Performer Viewer       | • Send operations  
  |                           |                                                                          |                        | • Manage routines  |
| Manage Devices            | Allows the user to manage and configure the devices which belong to the system. | Performer Viewer       | • Include devices in the system  
  |                           |                                                                          | Administrator          | • Remove devices from system  
  |                           |                                                                          |                        | • Manage categories  
  |                           |                                                                          |                        | • Include devices to categories  
  |                           |                                                                          |                        | • Assign a location  |
| Manage Wait for Signal    | Allows the user to connect the terminal to the network and start listening to the devices of that network. | User                   | • Activate wait for signal  
  |                           |                                                                          |                        | • Deactivate wait for signal  |
| Manage Notifications      | Allows the user to receive the notifications coming from different devices directly or through an external system. | Viewer                 | • Visualize notification  
  |                           |                                                                          |                        | • Save notification  
  |                           |                                                                          |                        | • Delete notification  |
| Search Devices            | Allows the user to search for devices already registered to the system, belonging to the local or to external networks. | Administrator Performer Viewer | • Search a specific device  
  |                           |                                                                          |                        | • Search devices by category  
  |                           |                                                                          |                        | • Search devices by criteria  |
| Store Information         | Defines how the system can store the information gathered by different devices about the environment or the state of the devices. | Administrator Performer Viewer | • Store locally the information  
  |                           |                                                                          |                        | • Store externally the info  
  |                           |                                                                          |                        | • Store in a device the info  |
| Retrieve Stored Information | Allows the user to retrieve information stored in the terminal or in an external system. | Administrator Performer Viewer | • Local information retrieval  
  |                           |                                                                          |                        | • External information retrieval  |
| Get Information from      | Allows the user to request information to the devices of the network.       | Administrator Performer Viewer | • Get information from devices associated to the application  
  | Devices                   |                                                                          |                        | • Get information from devices associated to the external system  
  |                           |                                                                          |                        | • Get information from devices of the same network  |
| Visualize Information     | Allows the user to visualize the information related to or produced by devices in different ways. | Administrator Performer Viewer | • Display information  |
| Share Information         | Allows the user to share information through a communication channel with other users or systems. | Administrator Performer Viewer | • Share information  |
6.2. IoT Integration

![Diagram showing main use cases for IoT applications]

**Figure 6.14:** Main use cases for the IoT applications.

...[Diagram showing events]

**Figure 6.15:** IoT Extensions: Events.

**Action Events**

This category regroups two types of events: *Timer event*, denoting the time on which the associated action is scheduled for execution; and *Repeat event*, specifying the time on which the execution of the associated action will be repeated. We have defined a new class for each type of those events: `TimerEvent` and `RepeatEvent`.

**6.2.2 IoT Actions**

This category contains the components allowing the modeling of the actions triggered when the user interacts with different IoT devices. Those actions can be grouped into two categories: *Device actions*, that represent the actions sent directly to the devices; and *Intermediary actions*, that represent the actions sent to the devices through an *Intermediary* (a component that manages the communication between the user and the devices). Each category can be further decomposed into two subcategories: *Set* and *Get*...
actions.

Set Actions

This category contains the actions which permit the user to send to one or more devices, a series of identifiers of the operations or programs which those devices have to perform or execute. We assume that the operations are known a priori by the devices, thus when we send an identifier of an operation to a given device, the device knows how to perform the corresponding operation. The Set operations are mainly used to configure the devices (e.g.: change the range in which the sensors are activated) and to perform specific actions such as turn on and turn off the device.

Get Actions

The Get actions are mainly used to retrieve the information from devices, category of devices, a program or an operation. We have defined a new class, GetAction, that allows the modeling of those actions. The class Get has been further extended to represent the specific data to retrieve. Examples of those data include details and state of the devices.
Figure 6.19: Example of IoT-based application modeling. The model allows the user to select one device from a list of available devices, visualize the details of the selected device, send a task to be executed.

device, information provided by the device and status of the operation assigned to the device. For the previous actions, we assume that the devices execute specified operations once the user triggers the action. But there exist other cases in which the user wants to schedule the execution of a given action at a specific time.

**Plan Action**

We have defined a specific action, called *Plan*, to model those operations which are not executed immediately by the devices but scheduled for execution (once or several times) in a subsequent moment. *Plan* is an asynchronous action that waits until the time scheduled for the execution of the operation. It inputs the targeted devices, execution time, operations, and optionally (for the repeating actions or operations) the number of repetitions.

### 6.2.3 Modeling Example

This section presents a piece of the interaction model allowing the user to interact with devices of a smart-house. The model in [6.19](#) corresponds to the piece of the application which allows the user to increase the intensity of a *smart lamp*. In this simplified example, the interactions start with a selection of the concerned lamp from a list of available devices. Then, a screen containing the details (thumbnail, name, state, intensity and a list of allowed operations) of the selected lamp is displayed to the user. Finally, the user chooses one operation from the list of operations associated to that lamp.
Chapter 6. User Interaction Modeling

6.3 Design Methodology

Front-end design of mobile apps is a complex task, the content and the navigation among them must be well designed at the purpose of exploiting at best the limited space available. Providing design patterns for both the content organization and navigation could help application designers to find solutions to common design challenges and to reuse them. In this section, we present a set of design patterns both for UI modeling for mobile and IoT-based applications to illustrate and simplify the modeling of those applications using the defined methodology and the tools. Moreover we show the matching between the IoT patterns and the user interface patterns defined in the context of IFML [25]. Furthermore we present the data synchronization patterns and we analyze their compatibility with the user interaction patterns for IoT-based applications.

6.3.1 User Interaction Patterns

The work on [25] introduced a set of design patterns that can be used to address typical issues (related to interface organization, content and navigation) of user interface modeling in general. We report in Table 6.2 a subset of those patterns we used as building blocks to model the user interactions patterns for the IoT applications. Table 6.7 shows a matching between those UI patterns with the IoT patterns defined in Section 6.3.3. As rows of Table 6.7, we list the IoT patterns, while as columns we have generic user interaction patterns. A checked cell \((i \times j)\) means that the \(j\)th UI pattern has been (can be) used to model the \(i\)th IoT pattern.

CRUD Patterns

The work on [25] covered also the traditional database operations of creation, update and deletion of an object of a given class (entity). In the IoT context, those patterns are used to configure an IoT system by adding, updating or removing an IoT object (such as IoT device, device categories, and intermediary elements) to the repository of the system. Those patterns are not explicitly considered in this section since they regard the static part of the IoT system. In the rest of this document we assume that the system is already configured.

Permission and Access Configuration

Security is a key issue in IoT systems. In Section 6.2, we have reported the common roles and corresponding access rights in an IoT system. The configuration of permission and access rights is done by using the CRUD patterns on users, groups (teams) and by assigning the access rights to the group of users. A piece of a user interface showing such configuration is shown in Figure 6.20 while an IFML model showing a complete example of the access configuration is reported in Figure 6.21.

6.3.2 Data Synchronization Patterns

There are several factors to consider when trying building a model to describe data alignment. The work on [77] introduced a set of data synchronization patterns that can be applied in the context of IoT-based applications. We report those patterns in Table 6.3 while a synthesis of the compatibility between those patterns and user interaction patterns
6.3. Design Methodology

Figure 6.20: A piece of a mock-up exemplifying the configuration of permissions and access rules.

Figure 6.21: A piece of IFML model exemplifying the configuration of permissions and access rules.
### Table 6.2: User Interaction Patterns.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Details and Multi-Details</td>
<td>Present some items and a selection permits the user to access the details of one instance at a time.</td>
</tr>
<tr>
<td>Multi-level Master Details</td>
<td>Also called cascaded index, consists of a sequence of lists over distinct classes, such that each List specifies a change of focus from one object, selected from the index, to the set of objects related to it via an association role. In the end, a single object is shown.</td>
</tr>
<tr>
<td>Default Selection</td>
<td>Simulates a user’s choice at the initial access of a list, thus selecting a default instance.</td>
</tr>
<tr>
<td>Multi-field Form</td>
<td>Form for submitting information through several fields.</td>
</tr>
<tr>
<td>Preloaded Field</td>
<td>Variant of Multi-field Form where some fields are preloaded with existing values.</td>
</tr>
<tr>
<td>Pre-assigned Selection Field</td>
<td>Form where the value of a selection field is pre-selected.</td>
</tr>
<tr>
<td>Data Lookup</td>
<td>Useful for data entry task that involves a complex form with choices among many options, such as in the case of form filling with large product catalogs.</td>
</tr>
<tr>
<td>Cascade Selection Fields</td>
<td>Useful for data entry task that involves entering a set of selections which have some kind of dependency between each others.</td>
</tr>
<tr>
<td>Basic Search</td>
<td>Keyword search upon a collection of items.</td>
</tr>
<tr>
<td>Location-aware Search</td>
<td>Enables search of items that are related and close to the current user position.</td>
</tr>
<tr>
<td>Login</td>
<td>Recognizes and checks for validity a user-provided identity.</td>
</tr>
<tr>
<td>User profile display and manage-ment</td>
<td>Shows and enables editing application-dependent information associated with the identity of an authenticated user.</td>
</tr>
</tbody>
</table>
6.3. Design Methodology

Table 6.3: Data Synchronization Patterns.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Data Synchronizations</td>
<td>Managing a data synchronization event asynchronously and without blocking the user interface.</td>
</tr>
<tr>
<td>Synchronous Data Synchronization</td>
<td>Manage a data synchronization event synchronously; blocking the user interface while it occurs.</td>
</tr>
<tr>
<td>Partial Storage</td>
<td>Synchronize and store data only as needed to optimize network bandwidth and storage space usage.</td>
</tr>
<tr>
<td>Complete Storage</td>
<td>Synchronize and store data before it is needed so the application has better response or loading time.</td>
</tr>
<tr>
<td>Full Transfer</td>
<td>On a synchronization event, the entire dataset is transferred between the mobile device and the remote system.</td>
</tr>
<tr>
<td>Timestamp Transfer</td>
<td>On a synchronization event, only the parts of the dataset changed since the last synchronization are transferred between the mobile device and the remote system using a last-changed timestamp.</td>
</tr>
<tr>
<td>Mathematical Transfer</td>
<td>On a synchronization event, only the parts of the dataset changed since the last synchronization are transferred between the mobile device and the remote system using a mathematical method.</td>
</tr>
</tbody>
</table>

for the IoT-based applications is reported in Table 6.8. Where we list the user interaction patterns for the IoT-based applications as rows and the data synchronization patterns as columns. A checked cell indicates a possible match in the adoption of the corresponding pair of patterns.

6.3.3 IoT Patterns

In this section we introduce a set of patterns that can be used to tackle typical problems in the design of the user interaction with the IoT systems. We group those patterns in three categories: (i) Set Patterns. This category regroups patterns that allow the user to send to the devices operations or programs to be executed. Those patterns are reported in Table 6.4; (ii) Get Patterns, which contains the interaction patterns that allow the user to retrieve information from a device, category, program or an operation (reported in Table 6.5); and (iii) Event-based Patterns reported in Table 6.6.
### Table 6.4: IoT User Interaction Patterns: Set Patterns.

<table>
<thead>
<tr>
<th>ID</th>
<th>Pattern</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>One Device One Operation</td>
<td>This pattern allows the user to set an operation to be executed by one specific device. The user selects a device of interest from a list of the devices of the system. Then, he chooses the operation to be performed from a list of operations supported by the selected device.</td>
<td><img src="image1" alt="Diagram P1" /></td>
</tr>
<tr>
<td>P2</td>
<td>One Device More Operations</td>
<td>This pattern allows the user to send to a single device a set of the operations to be performed. The interactions start with the selection of a device of interest. Then the user selects desired operations from a list of supported operations.</td>
<td><img src="image2" alt="Diagram P2" /></td>
</tr>
<tr>
<td>P3</td>
<td>More Devices One Operation</td>
<td>This pattern allows the user to send to many devices one operation to be executed. The interactions start by selecting the devices of interest. Then the user selects an operation (from a list of the operations supported by the selected devices) to be executed by those devices.</td>
<td><img src="image3" alt="Diagram P3" /></td>
</tr>
<tr>
<td>P4</td>
<td>More Devices More Operations</td>
<td>This pattern allows the user to send a set of operations to different devices. Those operations are not necessary the same for all devices, thus the operations must be binded to the devices which can perform them.</td>
<td><img src="image4" alt="Diagram P4" /></td>
</tr>
<tr>
<td>P5</td>
<td>One Device One Program</td>
<td>This pattern allows the user to send the program (identifier) to the device which will execute it. A program is a set of operations which have to be executed in a precise order. We assume that the programs are already configured in the devices, thus, the user has only to send the program identifier to the device.</td>
<td><img src="image5" alt="Diagram P5" /></td>
</tr>
<tr>
<td>P6</td>
<td>One Category More Operations</td>
<td>This pattern allows the user to set operations to different devices based on the groups they belong to, without needing to select one device at a time.</td>
<td><img src="image6" alt="Diagram P6" /></td>
</tr>
</tbody>
</table>
### 6.3. Design Methodology

**Table 6.5: IoT User Interaction Patterns: Get Patterns.**

<table>
<thead>
<tr>
<th>ID</th>
<th>Pattern</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>P7</td>
<td>Get Details of a Device</td>
<td>The user retrieves the general information about the device such as Id, name, description, and model. The user selects a device he is interested in from a list of devices.</td>
<td>![Diagram for P7]</td>
</tr>
<tr>
<td>P8</td>
<td>Get State of the Device</td>
<td>This pattern allows the user to retrieve the current state of a given device. The interactions start with the selection of the device for which the user needs to know the state. Then, the corresponding state is displayed to the user.</td>
<td>![Diagram for P8]</td>
</tr>
<tr>
<td>P9</td>
<td>Get Information from the Device</td>
<td>This pattern allows the user to retrieve the information provided by a device about the monitored object. The interactions start with the selection of the device for which the user needs to know the information of the monitored object. Then, the requested information is displayed to the user.</td>
<td>![Diagram for P9]</td>
</tr>
<tr>
<td>P10</td>
<td>Get Information for One Category</td>
<td>This pattern allows the user to get the information from all devices of the same category.</td>
<td>![Diagram for P10]</td>
</tr>
<tr>
<td>P11</td>
<td>Search Device</td>
<td>This pattern allows the user to search a specific device. The search of the device can be done in different ways depending on the application and on the devices.</td>
<td>![Diagram for P11]</td>
</tr>
<tr>
<td>P12</td>
<td>Nearby Devices</td>
<td>This pattern allows the user to retrieve all the devices near to a given location. The location can be set by the user or retrieved from the ContextDimension, Position, which represents the location information of the device used to access the application.</td>
<td>![Diagram for P12]</td>
</tr>
</tbody>
</table>
### Table 6.6: IoT User Interaction Patterns: Event-based Patterns.

<table>
<thead>
<tr>
<th>ID</th>
<th>Pattern</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>P13</td>
<td>Pull Information</td>
<td>This pattern allows the user to check periodically availability of new data from devices. To save some resources like power, for the data that can be delayed for some amount of time without impacting on the outcome of the application, the user can decide to activate periodically the listening service and pull all the information from the devices.</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td>P14</td>
<td>Application Launch</td>
<td>This pattern allows the user to retrieve the information sent by the devices when the application was not running or when he was offline. The launching event calls the external system and gets the notifications sent by all the devices when the user was offline.</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
<tr>
<td>P15</td>
<td>Push Information</td>
<td>This pattern allows the user to visualize the messages sent by an IoT device as a push notification.</td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
</tbody>
</table>

### IoT Patterns

<table>
<thead>
<tr>
<th>User Interaction Patterns</th>
<th>Master Details</th>
<th>Multi-level Master Details</th>
<th>Default Selection</th>
<th>Multi-field Form</th>
<th>Preloaded Field</th>
<th>Data Look up</th>
<th>Cascade Selection Fields</th>
<th>Basic Search</th>
<th>Location-aware Search</th>
<th>Login</th>
<th>User Profile Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Device One Operation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Device More Operations</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>More Devices One Operation</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>More Devices More Operations</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Device One Program</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Category More Operations</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Details of a Device</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get State of the Device</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Information from the Device</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get Information for One Category</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search Device</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nearby Devices</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Information</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Launch</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pull Information</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 6.7: Synthesis of User Interaction Patterns used to model IoT Patterns.

82
### 6.3. Design Methodology

#### IoT Patterns

<table>
<thead>
<tr>
<th>IoT Patterns</th>
<th>Data Synchronization Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Device One Operation</td>
<td>✓</td>
</tr>
<tr>
<td>One Device More Operations</td>
<td>✓</td>
</tr>
<tr>
<td>More Devices One Operation</td>
<td>✓</td>
</tr>
<tr>
<td>More Devices More Operations</td>
<td>✓</td>
</tr>
<tr>
<td>One Device One Program</td>
<td>✓</td>
</tr>
<tr>
<td>One Category More Operations</td>
<td>✓</td>
</tr>
<tr>
<td>Get Details of a Device</td>
<td>✓</td>
</tr>
<tr>
<td>Get State of the Device</td>
<td>✓</td>
</tr>
<tr>
<td>Get Information from the Device</td>
<td>✓</td>
</tr>
<tr>
<td>Get Information for One Category</td>
<td>✓</td>
</tr>
<tr>
<td>Search Device</td>
<td>✓</td>
</tr>
<tr>
<td>Nearby Devices</td>
<td>✓</td>
</tr>
<tr>
<td>Pull Information</td>
<td>✓</td>
</tr>
<tr>
<td>Application Launch</td>
<td>✓</td>
</tr>
<tr>
<td>Push Information</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Table 6.8: Synthesis of the compatibility between Data Synchronization Patterns and the IoT Patterns.*
Chapter 6. User Interaction Modeling

6.3.4 Running Example

To demonstrate the effectiveness of the designed extensions and usage of UI design patterns presented in Section 6.3.3, we have modeled the interaction of smart-home, an application that allows a user to interact with different devices of a smart home system. Figure 6.22 contains a piece of the user interface of smart-home application. This UI is divided in following navigation paths:

i. Manage cameras. When the user selects manage camera from the Home screen, a new screen Cameras showing a list of available cameras is displayed. The button Details associated to each camera allows the user to access to the details, state and current image, of the selected camera;

ii. Manage Lights. Once the user selects Manage Lights from the Home screen, a new screen called Lights is displayed. The screen Lights, contains a list of available lights with their current state (ON or OFF). The user can change the state of the selected light by pressing on/of button associated to each light;

Figure 6.22: Case study: a piece of user interface of smart-house application.
iii. Manage Alarms. The path which allows the user to see the logs of recent alarms. Once the user selects manage alarm from the home screen, a new screen Recent Alarms containing a list of the recent alarms is displayed.

Figure 6.23 shows the IFML model describing the user interaction of the piece of smart-home application presented in Figure 6.22. The interaction model is obtained by combining the following IoT user interaction patterns:

- Get Information from whole Category, used to retrieve the current status (on/off) of the monitored lights;
- Get State of the Device, used to retrieve the current state of Camera01;
- Get Information from the Device, used to retrieve the information about the object monitored (the image displayed in the screen Camera 01) by the Camera01;
- One Device One Operation, used for instance to turn off the Light01;
- Get Details of a Device, used to access the details of the selected logLine of the alarms. The details of the IoT devices can be also retrieved from the repository of the system;
- Store Information, used to store the new alarm;
- Push Information, used to inform the user about the new alarm. Notice that in the showed case, the new alarm arrived (as a notification message) when the user was visualizing updated list Lights after turning off the Light01.

### 6.3.5 Modeling Tool

A snapshot of the modeling interface is shown in Figure 6.25. The tool has been implemented using the Obeo Sirius framework (http://www.eclipse.org/sirius/) and will be released as opensource Eclipse Project, and will benefit from the contribution of the community of MDE developers.
Chapter 6. User Interaction Modeling

Figure 6.24: Example of a real case implementation

Figure 6.25: Snapshot of the opensource IFML modeling tool developed in this research.
CHAPTER 7

Automatic Code Generation

This chapter presents different approaches to generate the code for mobile applications from the models describing those applications. It reports the results of a comparative study conducted to identify the best trade-offs between various automatic code generation approaches. The results presented in this chapter are described through a concrete case: the identification of the best code generation strategy for WebRatio, a software development company, interested in producing a MDD tool for designing and developing mobile applications to enter the mobile apps market.

The rest of this paper is organized as follows: Section 7.1 overviews various code generation strategies; Section 7.2 shows a running example; Section 7.3 presents the code generation options; Section 7.4 compares the presented code generation strategies; and the Section 7.5 concludes.

7.1 Introduction

Software development organizations that aim to exploit the mobile market have plenty of new business opportunities through direct sales in app stores and delivery of B2E and B2B solutions. Nevertheless, the vastness and diversity of mobile devices and operating systems available on the market oblige companies to produce and deploy the same app several times, once for each of the different mobile platforms. Unfortunately, cross-platform and multi-device development is a barrier for today’s IT solution providers, especially SMEs, due to the high cost and technical complexity of targeting development to a wide spectrum of devices, which differ in format, interaction paradigm, and software architecture. The adoption of model-driven development (MDD) can simplify multi-device development, reducing substantially cost and development time, so as to increase the profit of SME solution providers and at the same time reduce the price and total cost of ownership for end-customers.
When following a MDD approach, several code-generation strategies are possible depending, both, on the abstraction level to be used when modeling the application and the abstraction level of the code to be generated. I use the model-driven architecture (MDA) as a reference framework to illustrate the different alternatives.

This scenario is faced by many companies today and given the variety of strategies, choosing the wrong one can have dramatic consequences for the company. Roughly speaking a company should choose among the following general options (Figure 7.1):

1. **PIM-to-Native Code (NC)**. This option states that the app code is generated from a PIM describing the app. Cross platform is achieved by providing one native code generator for each targeted platform;

2. **PIM-to-PSM-to-NC**. A global PIM is transformed into a set of PSMs that refine it for specific platforms. These lower-level PSM models are the input of the code-generator for the corresponding platforms;

3. **PSM-to-NC**. It consists on defining directly the PSMs, one per each development platform and then generate the app code from these PSMs;

4. **PIM-to-Cross Platform Code (CPC)**. This option takes as input the platform independent model and generates the code required by the cross platform framework (such as PhoneGap, AppCelerator Titanium, and Xamarin) to produce the cross platform apps; and

5. **PIM-to-Framework Specific Model (FSM)-to-CPC**. With respect to the PIM-to-CPC option, this approach introduces the FSM which gathers the information regarding the cross platform framework used to produce the apps. FSM is a PSM in which the Platform in the MDA terminology, is actually a Cross-Platform Framework for mobile apps development.

I present the results of a comparative study conducted to identify the best trade-offs between those alternatives. The study included the development of the code generators for native platforms (Android and iOS) and generators for cross-platform frameworks.
7.2. Applied Use Case

This section presents the app, *Movies Manager*, which is used to illustrate the different code generation approaches discussed in next sections. The domain model is specified in the UML class diagram while the user interaction is modeled using the Mobile IFML language.

7.2.1 The Movies Manager App Running Example

Movies Manager is a simple mobile app which keeps track of the movies watched by the user. The app allows the user to access a list of available movies, see the details of a movie, add a new movie, and finally delete a movie from the app.

![Figure 7.2: A piece of Mobile IFML model (PIM) describing Movies Manager app. It models the listing, the adding, the detail visualization and the editing of a movie. The model has been specified using the IFML editor (http://ifml.github.io/)](image-url)
Chapter 7. Automatic Code Generation

Figure 7.3: The UI of Movies Manager app (Android version): the home screen, which displays the list of movies, is shown on the left side. The screen in the middle allows the adding of a new movie while the last screen displays the details of a movie.

The Domain Model of the Movies Manager app would be just a simple Movie class with the thumbnail, the title, the year of publication, and its genre as properties. Therefore, Figure 7.2 only shows the Mobile IFML model describing the content and the interaction flows of the app. The model specifying the app has four screens:

1. Movie List screen. Is the home screen of the app. It contains a List, Movie List, which displays the thumbnail and the title for each movie managed by the app. From it, a user can access the Details of a selected movie, via details event, or add a new movie by following the add movie event;

2. Add Movie screen. It allows the user to add a new movie. It contains an entry Form, AddMovieForm, allowing the user to provide the details of the movie. The Long press event, Take photo, associated to the field Thumbnail allows the user to take a picture to use as the thumbnail for the movie he is entering. The Take photo event triggers a Photo Camera action, shoot. The Save event associated to the form allows user to submit the information to the system by invoking AddMovie Action;

3. Movie Details screen. It displays the details of a selected a movie. It is associated with two events allowing the deletion and editing of a movie; and

4. Edit Movie screen which allows the user to modify the information of the movie.

Figure 7.3 shows a piece of the UI of Movies Manager app generated from the models described in this section.
7.3. Code Generators

This section describes in more detail the different approaches for the automatic code generation for mobile apps depicted in the introduction.

7.3.1 PIM-to-NC: Generation of the Native Apps from PIM

The PIM-to-NC approach, which states that the app code is generated from the platform independent model, is described in this section through an example generating the Android native code for the Movie Manager app.

An Android App is written in Java. The Android SDK tools compile the Java code into an APK (Android package), which is an archive file with an .apk extension. The APK file is used by the android-powered devices to install the app. The essential building blocks of an Android app are called App Components. They are of four types: activities, services, content providers, and broadcast receivers. Each component has a distinct life-cycle and plays a specific role to define the behavior of the app. The Android generator depicted in figure 7.5 consists of a set of Acceleo\(^1\) templates which take the app’s models in input and generate the app code. The generated code consists of Java Classes implementing the app behavior and the XML files managing the app’s layout.

7.3.2 PIM-to-PSM-to-NC: Native Apps from PIM through PSM

The PIM-to-PSM-to-NC approach consists firstly in the specification of the app independently to the platform that will be used to implement the executable code and, in a second step, on creating different PSMs (one for each platform). A PSM is a model refined and annotated based on the specific characteristics of the platform to maximize

---

\(^1\)http://www.acceleo.org
the quality of the generated code. The running code is generated from those PSMs. Figure 7.4 shows a piece of the metamodel of the PSM language, Mobile IFML for iOS, designed as an extension of Mobile IFML to model the specific information regarding the behavior and structure of the app on iOS Platform. Similar PSM languages would be required for the other platforms. The iOS extensions address:

- **Events.** The iOS platform supports a large set of gestures and provides rich APIs to implement the related events. The mobile IFML events have been extended to better reflect the events managed in iOS; 7.4(a) reports a fragment of this extension that details the Platform Independent TouchEvent into iOS specific PinchEvent, RotateEvent and PanEvent which directly map to the iOS classes UIPinchGestureRecognizer, UIRotationGestureRecognizer, and UIPanGestureRecognizer respectively.

- **View components.** The iOS platform provides many types of views to help present and organize app’s content. The mobile IFML MobileViewComponent have been extended to better implement the iOS views; 7.4(b) reports a fragment of this extension that details the Platform Independent MobileViewComponent into iOS specific WebViewComponent, AdvertisingViewComponent, and SearchViewComponent which directly map to the iOS classes WebView, AdBannerView and SearchView respectively.

- **Mobile context.** For each sensor managed by iOS platform, I defined a iOSContextVariable extending the MobileContextVariable Mobile IFML class, capturing the readings of device’s sensor.

The Code Generator for iOS depicted in figure 7.6 consists of a M2M and M2T transformations applied in sequence. The M2M, conceptualized through a set of ATL rules, is applied to the models describing the generic behavior of the app to produce a corresponding iOS model refined with iOS-specific information. The M2T transformation, conceptualized through a set of Xtend templates, is applied to that refined iOS model to generate the app code (Objective-C code).

### 7.3.3 PSM-to-NC: Native Apps directly from the PSMs

The code generation options presented in previous sections start from the PIMs and produce the app’s code by applying a chain of M2M and M2T transformations. Nevertheless, another option is to skip the PIM level, specially when targeting a single

---

2https://www.eclipse.org/atl/
7.3. Code Generators

![Diagram of PSM-to-NC: the app’s requirements are specified in a platform specific model from which the app code is generated.]

**Figure 7.7:** PSM-to-NC: the app’s requirements are specified in a platform specific model from which the app code is generated.

![Diagram of PIM-to-CPC: the process for code generation for PhoneGap. The app’s requirements are specified in Mobile IFML and UML models. The M2T rules are applied to those models to generate the files required by phonegap builder to produce the app files.]

**Figure 7.8:** PIM-to-CPC: the process for code generation for PhoneGap. The app’s requirements are specified in Mobile IFML and UML models. The M2T rules are applied to those models to generate the files required by phonegap builder to produce the app files.

The PSM-to-NC approach states that the app’s requirements are directly specified in a platform specific manner, i.e. the model describing the app contains all the information required to implement an executable code in a given development platform (Figure 7.7).

### 7.3.4 PIM-to-CPC: Generation of Cross-Platform Apps from PIM

Providing a native implementation might not be always preferred due to the development cost, in terms of resources and time associated with the development activity for each platform, even if this problematic is soften when using MDD-based approaches. Using a target which is already cross-platform per se, simplifies the options presented in previous paragraphs by reducing the time to market and the development costs at the cost of losing the complete control of the native part of the apps.

The PIM-to-CPC option, which generates the code for cross platform frameworks from a PIM is described through an example which generates the code for PhoneGap framework. PhoneGap is a web based mobile development framework, based on the open source Cordova project. It allows using the standard web technologies (HTML5, CSS3, and JS) for cross platform development. PhoneGap produces a binary app archive (ipa file for iOS, apk file for Android, xap file for Windows Phone, and so on) that can be distributed through standard app ecosystems such as iTunes Store, Android Market, Amazon Market, and Windows Phone Marketplace.

The PhoneGap generator depicted in figure 7.8 takes as input the app’s models and generates the JavaScript, HTML5 and CSS optimized for Phonegap framework. The
Chapter 7. Automatic Code Generation

Figure 7.9: PIM-to-FSM-to-CPC: the app’s requirements are initially specified in a PIM. The requirements are then refined in a FSM which specifies also the information about the targeted cross-platform framework. The code required by the framework to produce the app’s files (one for each deployment platform) is generated from that FSM.

generated code is wrapped in the Cordova container and then sent to the Build PhoneGap which produces the app files.

7.3.5 PIM-to-FSM-to-CPC: Cross-Platform Apps from PIMs through FSMs

The code for cross platform frameworks can be also generated from FSM (a PSM in which the Platform in MDA, is actually a Cross-Platform Framework for mobile apps development). The model describing the app contains in this case the specific information regarding the behavior and the structure of the app on a specific cross platform framework. Figure 7.9 shows the approach which generates the apps’ code from a FSM derived from a PIM. The same result can be reached by using directly (without passing through PIM level if desired) a FSM allowing the specification of the app in terms of the targeted cross platform framework.

7.4 Discussion

This section compares the code generation approaches presented in previous sections and provides the general guidelines for selecting the winning approach for a specific scenario. It ends by presenting a concrete case in which the guidelines have been applied to select the best approach for WebRatio based on the requirements and the expertise of the company.

7.4.1 Effort Analysis for the Different Approaches

Table 7.1 provides a summary of the effort involved in the development of the needed languages and transformations for each approach. For the purpose of my analysis, it doesn’t make sense to mention the exact number of days to complete each task since the exact effort is context (expertise of developer, tools, etc.) dependent. Based on previous experience, I classify the effort in three categories: Low, Medium and High. Where Low represents the effort less than 10 man-days, Medium represents effort between 10 and 25 man-days while High corresponds to more than 25 man-days effort. For the PIM language, the effort depends on whether the language is designed from scratch or as an extension of an existing standard. Based on my experience in developing a
new PIM-level language (the IFML standard [26]) and its mobile extension (the Mobile IFML [27]), developing a new PIM language is a costly task while extending an existing standard requires low effort.

<table>
<thead>
<tr>
<th>Approach</th>
<th>PIM</th>
<th>PSM</th>
<th>MTM</th>
<th>M2T</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIM-to-NC</td>
<td>H</td>
<td>L</td>
<td>-</td>
<td>H</td>
</tr>
<tr>
<td>PIM-to-PSM-to-NC</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>PSM-to-NC</td>
<td>-</td>
<td>-</td>
<td>H</td>
<td>-</td>
</tr>
<tr>
<td>PIM-to-CPC</td>
<td>H</td>
<td>L</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PIM-to-FSM-to-CPC</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>H</td>
</tr>
</tbody>
</table>

Table 7.1: Effort analysis for code generation approaches. Each item is denoted with level of effort involved: low (L), medium (M), and high (H).

The tests of all the five alternatives, considered in this research, showed that there is no approach better than others in absolute terms but allowed us to make the following classification of the best scenarios per alternative:

1. PIM-to-NC. In this option, all the platform-specific details are embedded in the code generator. That makes this approach faster (since there is no need to generate the PSM). But embedding all the details of the app in the code generator reduces its flexibility. Typical data-driven native apps would be better suited for this option.

2. PIM-to-PSM-to-NC. This approach could be better when having to develop native apps, with complex functionalities. In fact, the PSM level allows to benefit from the specificities of the platform at the level of models and thus allowing more tuning options with respect to option (1).

3. PSM-to-NC. This option is better when developing native apps for one specific platform.

4. PIM-to-CPC. This option is better when developing the apps in which cross platform richness is more important than high performance. The cross platform approaches are generally advisable for the companies with limited resources. Several researchers concur that it is unlikely that a single vendor will dominate the future mobile-centric world. Therefore, for these companies, it is better to loose in native functionalities to cover the entire market.

5. PIM-to-FSM-to-CPC. Similarly to the approach (4), this option is suitable when developing the apps in which cross-platform richness is more important than high performance. The framework specific model allows to fully benefit from the cross platform framework specificity at the level of models. The option (5) is preferable to the option (4) when having a working team with low experience in the targeted cross-platform framework since is much more easier to control the app at model level than during the generation.

7.4.2 Industrial Experience: The Case of WebRatio

Webratio is an SME with a mature experience in model-driven development for web application. The company has around 60 engineers specialized in web technologies and
web application modeling using the IFML standard in a proprietary environment. The strong requirements for the company can be summarized as follows: (i) the tool shall produce the apps available for different platforms (initially iOS and Android shall be supported); (ii) the apps shall access the device’s common sensors; (iii) the apps shall be distributed through the standard apps’ ecosystems; and (iv) the typical customers of WebRatio are business users.

According to the classification of the best scenarios, the PIM-to-CPC and PIM-to-FSM-to-CPC approaches fit the requirements but the later requires an extra effort related to the PSM level which does not bring additional benefits to Webratio (since it is specialized in Web technologies). We advised the PIM-to-CPC approach, using PhoneGap as cross-platform development framework for the following main reasons:

- Cross-platform is more important than high performance. The company is not interested in full native apps;
- Typical customers are business users that privilege practical and usable UI with respect to fancy graphics;
- Heitkötter et al [58] advised PhoneGap as the best cross-platform framework when the native UI is not a strong requirement; and
- The team is experienced in app modeling and in standard Web technologies.

The approach advised to WebRatio is actually implemented in the company’s newly released product WebRatio Mobile Platform (released February 17, 2015).

7.5 Conclusion

This chapter presented different approaches to generate the code for mobile applications from the models describing those apps. It presented the results of a comparative study conducted to identify the best trade-off between different code generation strategies. This study showed that there is no approach better than others in absolute terms but provided some useful guidelines that helped us to identify the best strategy for the WebRatio company in particular. The approach advised to WebRatio is implemented in the company’s product, WebRatio Mobile Platform, released on February 17, 2015.
CHAPTER 8

User Behavior Analytics

This chapter presents a model-driven engineering approach that combines domain-specific languages used for user interaction design of the application, with further languages that describe different perspectives of the user tracking at runtime upon the resulting application. Merging information about the structure, content and semantics of the application with application usage data allows to enhance the understanding of how the users consume the provided content and thus has high potential of delivering valuable insights to designers and decision makers on the continuous improvement process of applications.

The proposed approach is based on an extended version of IFML focusing on Web application design, and on its implementation WebRatio [6,14], for the design and development of Web applications. I used web applications to illustrate the proposed approach since the web logs were already available for analysis. However, the illustrated results are of a general validity and can apply to any application developed using a model-driven development approach, given that the only requirements are conceptual models of the application and the runtime logs that allow the collection of log data containing the references to the conceptual model. In particular, the proposed approach applies to mobile and IoT-based applications developed following the development approach presented in previous chapters.

The remaining of this chapter is organized as follows: Section 8.1 introduces some background on Web analytics; Section 8.2 overviews the WebRatio implementation of IFML; Section 8.3 describes the proposed framework for merging logs, enriching them with information coming from models, and applying analysis over them; Section 8.4 gives the details of the implementation experience; and lastly Section 8.5 summarizes the results and concludes.
Chapter 8. User Behavior Analytics

8.1 Introduction

In recent years, the software language engineering community has put more and more emphasis on the design and experimentation of languages that cover the requirement specification [17,18], design [109], and verification/validation [28,37,41,42,53] of software artefacts. Some of these experiences have also spawned commercial products and thus have been applied in industrial settings, with excellent results.

On the other side, a completely different line of work (and research) is ongoing regarding the usage analysis of software applications (especially in the Web and mobile context). The aim of these researches is to extract analyses that lead to optimizing the user experience. This has become a customary step in taking informed decisions on how to improve user experience, which is nowadays crucial for the success of software applications. Indeed, with the increasing need to meet customer preferences and to understand customer behavior, Web analytics has become a very important contribution towards taking informed business and interaction decisions.

However, these problems are addressed by completely different techniques, namely: data mining, machine learning and big data analysis; or simpler pragmatic approaches in the user experience field, like A/B split testing. Actually, a variety of domain specific languages for data analysis and data processing pipeline specification exist (such as MatLab, R, and so on), which combine with modern big data analysis technologies (Hadoop, Hive, Spark, and so on), but they never relate back to the conceptualization of the application design addressed by software specification languages. Several tools exist that support analysis of Web server logs and extract information on application usage. However, those tools are unaware of the design structure and actual content managed by the application. Therefore, understanding the relation between the output of the quantitative log analysis and the structure of the application is not an easy task, especially for large and complex systems.

Furthermore, those tools provide analysis mainly at web page level and they do not take into account the content of the visited pages. The statistics on the users’ interaction with each single content of the visited web page can deepen the understanding of how the users consume the provided content and thus, help analysts and decision makers to take more informed decisions.

Some analytics tools, like Google Analytics [70], provide a way to track the user interactions at content level but at a high cost of hard coding which yields to high risk of errors and maintainability of the web application. Moreover, those event tracking mechanisms support a small subset of user interactions. The integration of appropriately designed modeling languages for user interaction development and Web logs has high potential of delivering valuable insights to designers and decision makers on the continuous improvement process of applications.

In this chapter I present a proposed model-driven user interaction analysis framework that integrates the application usage logs with the structural information of a Web application specification. The proposed framework contains a set of languages for the characterization of the applications at design time, as well as for the management of the runtime tracking through different logs of the application execution (namely, the runtime execution logs that track the behaviour of the system components, and the web server logs that track the requests received by the application from the user interface).
8.2. WebRatio Implementation of IFML

WebRatio Web Platform [6] is a model-driven development tool which implements the Web-extended version of IFML. The tool supports developers in the specification of the domain model and of the interaction flow model for web applications. The tool features model checking and full code generation that produces application code executable on top of any platform conforming to the J2EE specifications. Figure 8.1 represents an high level view of the WebRatio architecture, which supports developers at design time, code generation time, and execution time.

8.2.1 Design Time

WebRatio provides four main integrated modeling and development environments, supporting respectively: (i) The modeling of IFML diagrams for the specification of the user interaction; (ii) The modeling of UML class diagrams (or ER diagrams) for the content design; (iii) The graphical layout template and style design environment; and (iv) The specification and development of the modeling level and execution level of custom IFML elements defined as extensions of IFML. The WebRatio models are saved as XML documents.

8.2.2 Code Generation

Based on the input provided through the modeling and development environments, WebRatio provides code generators which transform the specifications of the application into concrete, executable implementations. The code generators are implemented using ANT, XSLT, and Groovy. The generated code consists of a Java EE code covering both front-end of back-end of web applications. The generated components include:

- The configuration file of the Controller which contains the navigation control flow logic.
- The action classes which are invoked by the Controller and, in turn, invoke the runtime services.
- The XML configuration files (called runtime descriptors) of the runtime services. In particular, every IFML ViewComponent in the model produces one runtime descriptor configuring the behavior of a generic runtime service. For example, a
specific List ViewComponent generates a runtime descriptor containing, among other things, the SQL code for extracting the content of the index from the data repository.

- The server-side templates for dynamically building the actual pages of the application.

### 8.2.3 Execution Time

The WebRatio run-time framework consists of object oriented components and services for organizing the business tier, clustered in three main layers: the service layer, the application layer, and the logging layer.

The service layer is deployed once and for all for every application and does not actually need code generation, because it comprises general purpose services. Therefore, at runtime one single service class is deployed for each type of component, which is then instantiated with the smart service creation approach described next. The deployed services are:

- a Page Service, that is in charge of calculating the whole page structure for every page of the site;
- a set of ViewComponent Services (one for each component type), in charge of executing the logic of each view component;
- a set of Action Services (one for each action type), in charge of executing the business logic of each IFML action;

These services get configured based on the XML configuration files (descriptors) allocated at the application layer: the code generator produces one JSP template and XML descriptor for each Page in the IFML model; and one XML descriptor for each IFML Action and ViewComponent. The execution logic features a *smart service creation* policy: services that implement ViewComponents or business actions are created upon request, cached, and reused across multiple requests, i.e., for every designed IFML element, which gets configured through the XML descriptors. Access to the information stored in the XML ViewComponent descriptors is granted by standard parsing tools. This behavior is tracked by the *Activity log*: a set of pre-built functions for logging each service is provided. The logging service can be extended with user-defined modules, so as to log the desired data.

In the proposed approach, I am more concerned with the *logging service*. It represents a valuable asset, as it allows the logging of the execution of all application objects and shows the queries executed to fulfill the HTTP page requests. This information, along with the HTTP access log data available on the Web server, and information from conceptual models are the inputs of the proposed user interaction analysis framework.

### 8.3 The User Interaction Analysis Framework

This section describes the proposed approach which aims at delivering a value added user interaction analyses from Web application usage by integrating: (i) the application logs, both in terms of Web server logs and runtime execution framework logs; (ii) the data from the conceptual models of the Web applications developed using a model-driven
8.3. The User Interaction Analysis Framework

Figure 8.2: The proposed user interaction analysis framework. The process starts by integrating the runtime data (Web Server access log and WebRatio runtime log) with the structural information gathered from the conceptual models of the application (content and interaction flow models); then it enriches the integrated logs with the database objects used to populate the requested pages. The third phase denormalizes the rich logs; and finally we perform the user interaction data analysis and visualize the results.

development approach; (iii) the data from the database objects used to populate the requested pages.

My approach is fully engineered around a set of languages devised to describe all the above resources. Therefore, I specify the languages for: extending IFML for the Web context; defining the grammar of the WebRatio runtime log and Web server logs; and defining the structure of the integrated, denormalized and enriched log.

The proposed user interaction analysis process relies on a set of steps that encompass: the integration of runtime data with conceptual model, enrichment of the integrated log with the database instances used to populated the requested pages, denormalization of rich log, analysis over the denormalized logs and results visualization, as shown in Figure 8.2. We apply model transformations for performing the various step of the process. I describe in detail the rationale behind each phase of the proposed approach in the next paragraphs, while the implementation details are reported in Section ??

8.3.1 Logs - Application Models Integration

This phase receives in input the application server logs, the WebRatio runtime logs, and the conceptual models describing the application. By integrating all this, it produces a single log which binds the information from application usage (runtime data) to the conceptual model of the application captured into interaction flow and content models.

Inputs

1. The WebRatio Runtime Log
   
   We defined a “verbose” version of the WebRatio runtime (RTX) log that stores events and data produced and consumed by the application runtime for serving page requests. It traces the history of the pages, view components, and operations that are executed, along with the executed queries. Moreover, the RTX log contains the data about the requested pages and their contents.
In order to have a clear picture of what is actually displayed in the pages, we rely on the IFML visibility conditions, i.e., rules computed at runtime for deciding whether parts of the interface are visible or not. The application designer can specify which components of a given page are displayed or hidden to the user based on some visibility conditions which are checked at runtime, and the RTX log contains for such component a boolean field which shows whether the component was displayed or not. Furthermore, all logged information contains a reference to the relevant application model elements. Each log line has the following structure (exemplified in Listing 8.1):

- The **Timestamp**, which is the time when the logged instruction is written in the log;
- The **Log Level**, which is the level to which the log line belongs;
- The **Host**, which is the host sending the request. In other words, the IP address of the computer used by the Web application User to navigate the application;
- The **Java class**, which is the java class executed;
- The **Session ID**, which is the User session id;
- The **Model element**, which is the model element executed;
- The **Message**, which is the log message.

**Listing 8.1:** Example of WebRatio Runtime log line.


2. **Content Model**

The content model specifies the organization of the persistence layer in terms of UML class diagram or ER diagram, where the core information is entities, attributes and relationships. More concretely, a content model is an XML file with all the information about the application data.

3. **Interaction Flow Model**

It is a Web-extended IFML model which describes the user interaction of the application. It models: (i) the composition of the Web application, in terms of web pages (ViewContainers) and widgets (ViewComponents); and (ii) the navigation paths between different pages and view components, through Web links.

**Output**

**Integrated Log** The result of this step is an enhanced Webratio Runtime Log in which each reference to the ViewComponents and ViewContainer is filled with the data corresponding to that element. Moreover, each ViewComponent is coupled with its DataBinding, showing which element of the content model the component shows (if any). An example resulting from the integration of Listing 8.1 is shown in Listing 8.2. The types of the elements are added (in the example: ViewContainer, Page, List) together with the reference to the content type that the component deals with (dataBinding=Book). Furthermore, all the elements that are not visible in the pages (visibility condition = false) are removed.
8.3. The User Interaction Analysis Framework

**Listing 8.2:** Example of Integrated Log line.

22 Jun 2016 11:10:51.761 [119354A67C7C0177D4A7F411E75BCDE7] [ViewContainer [Type : page, id : page21 ... ]] [List [id = pwu6Block][dataBinding = Book]]

### 8.3.2 Instance Filling

This phase enriches the integrated log produced in the previous phase with the database instances used to populate the requested content. The result of this phase is a *Rich Log* which contains all the information about the requested pages, the structure of each page, and the visited content.

**Inputs**

1. *The Application Server Log*
   
   The application server log is the common access log produced by the Web server/application server where the Web site is deployed. The access log records all the requests processed by the application server in the standard Extended Common Log Format. An example is shown in Listing 8.3.

   **Listing 8.3:** Example of Application Log line.

   0:0:0:0:0:0:0:1.
   

2. *Integrated Log*
   
   It is the log generated by the previous phase.

3. *Database.* Information about the database (e.g. the actual connection to the database) used to populate the requested pages.

**Output**

*Rich Log* The output of this phase is the *Rich Logs* (an example is shown in Listing 8.4), defined as enriched web logs that integrates the conventional HTTP data about pages requests with information about the components on each requested pages and the database instances used to populate the requested pages. For instance, in the example the details about the attributes of the visualized book are integrated (title, author, etc.). This allows to get a complete picture of what exactly the user is seeing in the page. I also compute which attributes are actually displayed or not, based on the ViewComponent configuration.

**Listing 8.4:** Example of Integrated Log line.

22 Jun 2016 11:10:51.761 [119354A67C7C0177D4A7F411E75BCDE7] [ViewContainer [Type: page, id : page21 ... ]] [List [id = pwu6Block][dataBinding = Book]][[title : "...", author : "...", ...]]

### 8.3.3 Log Denormalization

This phase receives the rich log produced in the previous phase and transforms it into a denormalized, column-based format expected by the analytics tool. The output of this phase is a *Denormalized Rich Log.*
Chapter 8. User Behavior Analytics

8.3.4 Analysis and Visualization

In this phase, the generated Denormalized Rich Log is feed into a big data analysis environment and a set of relevant data mining and visualization patterns are defined to allow investigation and exploration of the user interactions from the denormalized rich log.

8.4 Implementation Experience

This section reports on the implementation experience of the proposed approach. Before I jump to the implementation details of each phase described in Figure 8.2, I introduce the preliminary steps which produce the inputs to the first phase (Logs - Models Integration) of the framework. These preliminary steps are comprised in the Web application development within WebRatio: in particular, they are the modeling phase (which produces Content and Interaction Flow models); the code generation; and deployment, execution and monitoring of the generated application within the web and application server, which produces both the WebRatio Runtime Log and the access log.

After this steps are covered, the actual integration and analysis process can start.

8.4.1 Web Application Development in WebRatio

As described in Section 8.2, an executable Web application is generated by WebRatio from the interaction flow and content models by applying a set of transformation rules.

Web Application Modeling

This phase consists of application content modeling and interaction flow modeling.

Application Content Modeling. It produces the Content Model. The content model includes the Entities and the Relationships among them. Each Entity element has Attribute child element. The content model is stored in XML format;

Interaction Flow Modeling. It produces Interaction Flow Model. The interaction flow model is composed by linked pages. Each page contains various type of ViewComponents which display the page content. A subset of the ViewComponents of the Web-Extended version of IFML are reported in Table 8.1 where the Component ID column shows how corresponding component is referred in the log file. For example [pwu6...] in Listing 8.1 refers to a List component.

Code Generation

After the modeling phase, WebRatio generates the executable application code from the models describing the application.

Deployment, Execution, and Monitoring

In this phase the generated application is deployed on a Web and application server to let users interact with it. The WebRatio logger service logs all the interactions with the application and generates the WebRatio runtime log. The log file contains the event logs,
8.4. Implementation Experience

<table>
<thead>
<tr>
<th>ComponentID</th>
<th>IFML-Web ViewComponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>dau</td>
<td>Details</td>
</tr>
<tr>
<td>mdu</td>
<td>Multiple Details</td>
</tr>
<tr>
<td>inu</td>
<td>Simple List</td>
</tr>
<tr>
<td>pwu</td>
<td>List</td>
</tr>
<tr>
<td>hinu</td>
<td>Hierarchy</td>
</tr>
<tr>
<td>mciu</td>
<td>Checkable List</td>
</tr>
<tr>
<td>enu</td>
<td>Form</td>
</tr>
<tr>
<td>meu</td>
<td>Multiple Form</td>
</tr>
<tr>
<td>getu</td>
<td>View Component</td>
</tr>
<tr>
<td>ecu</td>
<td>Calendar</td>
</tr>
<tr>
<td>mssu</td>
<td>Message</td>
</tr>
</tbody>
</table>

Table 8.1: List of ViewComponent types defined in the Web-extended version of IFML, with the corresponding ComponentID used to reference them in the log files.

which give information on which pages, view component or links user has activated, as well as some logs on the state of the web project and more. Among these logs only the ones related to pages, ViewComponent and links are valuable in the current context.

8.4.2 Logs – Models Integration

This section describes how the integration of logs and model has been implemented. The xText\[^{46}\] has been used for defining the logs grammar and developing the extractors that create a model starting from the text file. The application models specified in IFML were then used for the actual enrichment phase, which was implemented as a model to model transformation in ATL\[^{2}\].

Log Language Design

I build the grammar of the WebRatio Runtime Log and the Integrated Log. The language design phase was carried out with the aim of making the subsequent model transformation easy to implement. Hence I identified three types of row in the WebRatio Runtime Log:

1. Creation: creation of a ViewComponent;
2. Query: execution of a database query;
3. Condition: evaluation of an ActivationExpression;

Listing 8.5: Fragment of the WebRatio Runtime Log Xtext grammar

```
RTXRow:
  timestamp=Timestamp  level=Level  thread=Thread
  package=Package  ‘-’  content=Content;

Content:
  ’{ ’  sessionId=ID ’} ’ ’{ ’  viewContainer=ViewContainer ’} ’
  ’{ ’  viewComponents+=InteractionFlowElement* ’} ’
  message=Expression
```

[^46]: https://eclipse.org/Xtext/
[^2]: https://eclipse.org/atl/
Chapter 8. User Behavior Analytics

Listing 8.5 shows a fragment of the Webratio Runtime Log grammar. As you can see a RTXRow (lines 1-3) is associated with a Content. A Content has an Expression (line 8) of different types (line 12). Then the following xText rules allow me to differentiate the types of row previously defined and to correctly parse the text file. For instance the Condition rule on lines 15-18 allows to extract the id and the result of an activation expression.

Listing 8.6: Fragment of the Integrated Log Xtext grammar.

Listing 8.6 shows a fragment of the Integrated Log. I kept only the information regarding the elements that are visualized (e.g., the ViewComponents) and I completed it with information regarding the instances that are shown (lines 20-28).
8.4. Implementation Experience

Model to Model Transformations

This phase receives in input the IFML and Content Model of the application and the WebRatio Runtime Log. The model to model transformation retrieves the information from the IFML model and adds the information regarding the ViewComponents by matching the ID in the model with the ID in the log.

Listing 8.7: Transformation rule for the first step of the transformation.

```
1 rule RXTRowToIntegratedRow {
2 from
3 s : WebRatio Runtime Log !RTXRow( 
4 s.isVisible() AND s.getType() = 'CREATION' 
5 )
6 to
7 t : IntegratedLog !IntegratedRow ( 
8 timeStamp <- s.timeStamp , 
9 content <- s.content 
10 )
11 }
12 }
13
14 rule RTXContentToEnrichedContent { 
15 from
16 s : WebRatio Runtime Log !Content 
17 to
18 t : IntegratedLog !Content ( 
19 sessionId <- s.sessionId , 
20 viewContainer <- s.viewContainer , 
21 viewComponents <- s.viewComponents 
22 )
23 }
24 }
25
26 rule RTXComponentToEnrichedComponent { 
27 from
28 s : WebRatio Runtime Log !ViewComponent 
29 to
30 t : IntegratedLog !ViewComponent ( 
31 id <- s.id , 
32 domainModel <-
33 thisModule.getDataBinding(s) 
34 )
35 }
36 }
```

Listing 8.7 shows the rules involved in the transformation. This transformation only considers the rows of type Creation (line 4) and that and use them to create new enriched rows in which the ViewComponent is associated with its DomainModel (line 31). In particular the function thisModule.getDataBinding(s) is a helper that given a ViewComponent it queries the IFML model and retrieves the correct DomainModel. Moreover isVisible() is an helper that looks in the Webratio Runtime Log for the Condition row related to the ViewContainer and checks its result. If visibility is false, the element is removed from the resulting integrated log.

Instances Filling

In this phase I add information regarding the actual instances that are visualized in the page. This final log is produced by integrating the Application Sever Log, the WebRatio
Chapter 8. User Behavior Analytics

Data: WebRatioRuntimeLog, ApplicationServerLog, IntegratedLog, Database

Result: RichLog

RichLog ← {};

for Row r in IntegratedLog do

if r.viewComponent.dataBinding!=NULL then

query ← getQuery(r.viewComponent, WebRatioRuntimeLog);
parameters ← getParameter(r.viewComponent, ApplicationServerLog);
query ← fillQuery(query, parameters);
resultSet ← executeQuery(query, Database);

if resultSet is not empty then

richRow ← r.append(parseResult(resultSet))
RichLog.append(richRow)

else

RichLog.append(integratedRow);
end

else

RichLog.append(integratedRow);
end

end

Algorithm 1: Instance filling algorithm.

Runtime Log, the IntegratedLog and the Application Database. The Application Sever Log is used for retrieving the information regarding the actual choice of the user (e.g. form field values, link clicked, etc..) while the WebRatio Runtime Log contains information about the query performed on the database. Together they allow to query the database in order to retrieve the objects that are visualized in a page.

The pseudocode listed in Algorithm 1 shows in details the steps performed for producing the Rich Log. The algorithm cycles over the row of the IntegratedLog (line 2). If a row refers to a ViewComponent that visualize some data (line 3) the algorithm retrieves the query to perform from the WebRatio Runtime Log (line 4), it gets the values of the query parameters from the Application Server Log (line 5) and uses them to complete the query (line 6). Finally the query is executed (line 7) and, if the result is not empty, a row is created by appending to the row of the integrated log the information regarding the instances retrieved.

8.4.3 Analysis and Visualization

This phase uses the big data analysis and visualization tools to derive information on the produced Rich Log. In the current implementation uses ElasticSearch, a distributed, scalable, and highly available real-time search and analytics engine, which serves as a back-end database for storing and querying large-scale enriched logs. Over that, I also use Kibana, a layer that provides querying, exploration and visualization capabilities over the data. Queries in Kibana are created through the user interface and the Elasticsearch request is automatically generated.

In this implementation I devised a set of standard queries and visualization that I deem useful in behaviour analysis. Further queries can be defined at will.
8.4. Implementation Experience

Figure 8.3: The Number of Pages Visited per User as a Percentage.

Figure 8.4: An Example Page Flow.

Visited Pages

Some insightful information related to the visited pages, similar to what other analytics tools provide are the following:

- The number of pages visited per certain time frame;
- The number of pages visited per certain time frame per user;
- The number of pages visited per certain time frame as a percentage;
- The number of pages visited per user as a percentage;
- The pages visited per certain time frame;
- The pages visited per user.
Chapter 8. User Behavior Analytics

Number of Users

It is possible to obtain the number of users per certain time frame which can be second, minute, hour, day, month, year or automatically assigned by Kibana. This is also a common information offered by many existing web analytics tools.

Page Flow

For each user, I can extract the page flow which shows how the user navigated the web site. For each page, it is possible to add which link is clicked and the target component of that link (Figure 8.5). In case the links are added, I can distinguish them as visible or invisible links. The visible links are the ones which are actually clicked by the user. The invisible links are called OK link or KO link in the WebRatio platform and denote the flow after the execution of associated action. The OK link is followed in case of success while the KO link is followed in case of failure of any action.

It is also possible to construct the page flow as in Google Analytics, using the data returned from the queries, as in the example given in Figure 8.4, however it is not possible to render it using the visualization capabilities of Kibana.

Link Flow

Various types of link flow can be obtained, here I report the significant ones:

- Link Flow per User (Link’s ViewComponent → Link → Link’s Target ViewComponent)
- Visible Link Flow per User (Link’s ViewComponent → Link → Link’s Target ViewComponent)
- Number of Links Clicked on Each Page
- Number of Visible Links Clicked on Each Page
- Links Followed per User
- Visible Links Followed per User

In Figure 8.5 and 8.6 two example visualizations are given for number of links clicked on each page together with link names and links followed for each user, ordered based on the timestamp.

ViewComponents

The number of a ViewComponent’s names fired per user and the number of a ViewComponent’s entities’ names fired per user can be visualized. This directly exploits the integration with the model and the database instances, and allows to understand what data items are visited most.

Display Attributes

The display attribute names of entities for each ViewComponent can also be displayed. The data/visualizations listed above which are per user can also be combined into all users by removing the aggregation on sessionId. Moreover, using the data that is obtained, it is possible to calculate any the statistical data. Combined with However, considering the visualization capabilities of Kibana, it is not always possible to display the results visually. In those cases, custom visualizations must be implemented.
8.4. Implementation Experience

To validate the approach and the implementation, I performed a set of experiments over an actual Web site. I followed the whole development, enrichment and analysis process. The experiments are based on a classical e-commerce web site, named BookStore. The application provides information on books such as their title, authors, price, category, description, reviews, comments and rating, and allows to navigate and purchase goods (the actual payment phase is not covered).

I run two experiments over it: the first one was run on a limited time-frame with a small number of live users (less than 10 at any moment), who were explicitly invited to join the experiment and were asked to navigate the site in the most natural and typical way possible. This allowed to collect a relevant number of user traces (sessions) and thus derive actual behaviors. However, due to limited time and effort available from users, the size of the experiment was not realistic with respect to large-scale real ecommerce web sites.

Therefore, I devised a second set of experiments, which focused instead on scalability and size, obtained by generating several synthetic experiments with randomized variations of the navigation paths of the first experiment, and simulating the navigation of a few thousands of users, that generated hundreds of sessions in parallel. This was done by instrumenting an automatic web site crawler that performed the navigation paths as instructed.

I run the approach in both settings and extracted queries and visualizations as described in the previous subsections. Scalability was challenging in the second setting: the input logs size was in the order of several hundreds megabytes (for instance, the experiment with 50 parallel sessions required around 240MB). Data processing took in the worst case more than 24h computation on a common PC (including model transformations and ElasticSearch indexing). However, after this was done, querying and visualization was very efficient.
Chapter 8. User Behavior Analytics

Figure 8.6: Links Followed per User (drill down and aggregation can be applied interactively in Kibana).

8.5 Conclusion

This chapter presented a model-driven specification and transformation approach that blends together design time information and runtime execution data from Web sites for generating insightful analyses of user behavior. As a limitation, this method cannot be applied to every existing web application, the web application must have been created using the WebRatio platform. A more general way to cover a wider range of web applications can be used through covering more modeling languages. Another point is that all the steps from analyzing to visualizing can be automatized to enable obtaining results easily, so that users who are not familiar with big data analysis technologies can obtain results easily.
CHAPTER 9

Conclusions and Future Work

9.1 Context and Relevance

9.1.1 Mobile Applications Development

Mobile devices constitute the most common computing device. A vast array of features has been incorporated into those devices to address the different demands of users spanning from games to serious business. This new computing model have brought intense competition and innovation among devices, OSs, and application providers who are continuously introducing increasingly powerful mobile devices and innovative operating systems into the market.

Even though the mobile OS market is beginning to mature and consolidate, most researches concur that it is unlikely that a single vendor will dominate the future mobile-centric world [58]. The dilemma between browser-based (HTML 5) and native (iOS, Android, and Windows Phone) interfaces remains relevant and will challenge the capacity of organizations to meet the increasing demand for mobile applications. Moreover, the vastness and diversity of mobile devices and operating systems available on the market oblige companies to produce and deploy the same application several times, once for each of the different mobile platforms.

Unfortunately, cross-platform and multi-device development is a barrier for today’s IT solution providers, especially SMEs, due to the high cost and technical complexity of targeting development to a wide spectrum of devices, which differ in format, interaction paradigm, and software architecture. Currently, there are several frameworks implementing different methodologies for cross platform application development (Web, Hybrid, Interpreted and Cross Compiled): examples include PhoneGap (Cordova), Appcelerator Titanium, and Xamarin. Nevertheless, these approaches still require manual programming which yields to high risks of errors, inconsistencies, and inefficiencies.
Chapter 9. Conclusions and Future Work

According to Gartner, by 2016 more than 300 billion applications will be downloaded annually. The number of apps that are available in the online markets has reached unseen numbers. In fact, by July 2015, the Google Play store counted 1.6 million of available apps while Apple’s App Store counted 1.5 million [4].

In parallel with these numbers, the market also expects an increase in the number of global smart-phones users, which is expected to surpass 2 billion by 2016 [1] in comparison with 1.4 billion users estimated in 2013 [4]. From those statistics we can expect a healthy market of mobile apps that would be powered by a steady increase in the number of mobile device users, which as of today have, on average, 41 apps installed on their devices [4].

Furthermore, the motivation of the software development companies to continue producing more and better apps is supported by recent industry figures, according to which global mobile app revenues are projected to surpass 76.52 billion U.S. dollars in 2017. ABI research forecasts in 2018, app revenues will be worth 92 billion U.S dollars [89].

9.1.2 IoT-based Applications Development

Currently, the IoT vision is mainly focused on the technological and infrastructure aspect, as well as on the management and analysis of the huge amount of generated data. In particular, so far the development of front-end and user interfaces for IoT systems has not played a relevant role in research. On the contrary, user interfaces must be a key part in the IoT ecosystem because they can play a key role in acceptance of solutions by final adopters. The intelligent things connected together can cooperate and exchange information, but their ultimate goal is to provide value to people. Such value can be perceived only through appropriate user interfaces, which visualize information (through dash-boards, reports, or infographics), let user navigate the information, and also interact with the sensors and devices, by setting properties, collecting data and regulating their behavior.

An increasing number of physical objects are being connected to the Internet at an unprecedented rate realizing the idea of the Internet of Things. Examples of such objects include thermostats and Heating, Ventilation, and Air Conditioning (HVAC) monitoring and control systems that enable smart homes. Moreover, the IoT offers a great market opportunity for equipment manufacturers, Internet service providers and application developers. The IoT smart objects are expected to reach 212 billion entities deployed globally by the end of 2020 [52]. Overall, the annual economic impact generated by the IoT is estimated to be in range of $2.7 trillion to $6.2 trillion by 2025 [45].

9.2 Main Contributions

This research proposed to face the challenges of mobile and IoT-based applications by exploiting abstraction, modeling and code generation in the spirit of model-driven development engineering. I covered all development phases, from design to execution, of mobile and IoT-based applications by applying model-driven development approach. The main contributions include: a modelling language and design methodology; code generators for the implementation phase; and a model-driven framework for user interaction analysis.
9.2. Main Contributions

9.2.1 Design Time

When following modeling driven development approach, the application is expressed through models from which the final code is obtained. In this research I defined a platform independent modeling language for mobile applications, Mobile IFML. It has been designed extending a OMG standard, Interaction Flow Modeling Language (IFML), as described in Section 6.1. Mobile IFML has been further extended to accommodate the IoT-specific requirements, as presented in Section 6.2. I defined a set of design patterns allowing the modeling of recurrent problems in user interaction modeling for both mobile and IoT-based applications, as discussed in Section 6.3.

9.2.2 Implementation

This research explored different approaches to generate the code for mobile applications from the models describing those applications. I conducted a comparative study to identify the best trade-off between different code generation strategies. As discussed in Chapter 7, this study showed that there is no approach better than others in absolute terms but provided some useful guidelines that can help to identify the best code generation strategy.

9.2.3 Execution Time

To enhance the understanding of how the users consume the provided content I proposed a model-driven engineering approach, described in Chapter 8. The proposed approach combines domain-specific languages, used for the user interaction design of the applications, with further languages that describe different perspectives of the user tracking at runtime upon the resulting application.

In parallel to the main focus of this thesis, I investigated on the supposed barriers to the adoption of model based development approaches. In, that context the issues of: modeling effort (many software practitioners consider model-based development approaches time-consuming, hence prefer to avoid using models which are believed as excessive, superfluous and unnecessary artifacts); and modeling language complexity and usability have been tackled. The next paragraph review those results.

9.2.4 Effective Modeling Effort in Model-based Development Approaches

Even though several studies provided evidence that the adoption of model-based approaches can significantly enhance both developers’ productivity and product quality, many software practitioners consider such approaches time-consuming, hence prefer to avoid using models which are believed as excessive, superfluous and unnecessary artifacts. As a result, software modeling is still not widely adopted in software development. I conducted a research with the aims of revealing whether it is actually modeling or designing that dominates the effort in the creative process of modeling software systems. The experiments, reported in Chapter 4 showed that more than 60% of the effort is spent in designing. For us, this means that the fault of supposedly unproductive processes should not be blamed on modeling, but to the (anyhow necessary) time devoted to thinking about the problem and identifying the solution.
Chapter 9. Conclusions and Future Work

9.2.5 Modeling Languages Design and Usability

I conducted a research on modeling language usability to understand how the designed languages are actually used and how they fit the users’ need. This investigation showed that available languages are either too complex (e.g.: BPMN) with respect to user need or do not exactly fit the domain (e.g.: UML). This happens because of the lack of right involvement of end-users in the language development process.

I proposed a user-centered approach allowing the adaptation of existing modeling languages to the user needs through a language simplification process. The proposed simplification process relies on: (i) the selection of the language elements to simplify, (ii) generation of a set of language variants for those elements, (iii) measurement of effectiveness of the variants through modeling sessions performed by end-users, and (iv) extraction of quantitative and qualitative data for guiding the selection of the best language refinement. The results of this study, as presented in Chapter 5, showed that the followed approach provides evidence and qualitative and quantitative data enabling the selection of the best language variant to be taken more objectively and in an informed way.

9.3 Future Directions

In the previous chapters, I discussed possible extensions of the presented contributions. In this section, I discuss in general the future directions derived from this work.

Language Simplification Future works will include the analysis of the concrete syntax (e.g., the graphical symbols to be used) as part of the simplification process, the application of the proposed approach when the modeling needs evolve (e.g., language maintenance and evolution) and, more importantly, the replication of this study in other domains to advance in the generalization of our language simplification process.

Dissecting Design Effort and Modeling Effort in MDD Approaches This research line can be extended in many directions. To increase the external validity, it would be interesting to obtain data from industrial projects about their design and modeling time. Another angle to enhance the generalizability of the presented findings is to study other types of models. This would enable us to study if the ratio of designing vs modeling depends on modeling language or not. Furthermore, the effects of modeling tools can be considered even though I do not expect a large influence of the tools. I analyzed different efforts considering only the design of new models. A complementary question would be to explore those effort considering model maintenance and evolution.

User Behavior Analytics Possible extensions of this research line include a study of additional Web-specific analyses (for instance, by combining the analysis with variant-based approaches such as A/B testing for checking how users react to different versions of the interface or of the rendered content); and exploration of different tools to enhance the visualizations.
Bibliography

Bibliography


Bibliography


