

Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Graphics and Interaction



**Automatic User Interface Generation  
Doctoral Thesis**

by

*Miroslav Macík*

A thesis submitted to  
the Faculty of Electrical Engineering, Czech Technical University in Prague,  
in partial fulfilment of the requirements for the degree of Doctor.

Ph.D. program: Electrical Engineering and Information Technology  
Branch of study: Information Science and Computer Engineering

Prague, January 2016

**Thesis Supervisor:**

prof. Ing. Pavel Slavík, CSc.  
Department of Computer Graphics and Interaction  
Faculty of Electrical Engineering  
Czech Technical University in Prague

Copyright © 2016 by Miroslav Macík

# Abstract

The development and broad public dissemination of modern information and communication technologies implies a need for new requirements to corresponding user interfaces. There is a significant rise of diversity in physical form-factors of interactive devices, as well as the underlying software platforms on which those devices are based. The vast majority of present society, including people with special needs, requires access to modern information technologies. This trend represents a challenge for corresponding user interface technologies. Corresponding technologies should offer a sufficient level of adaptivity and context-sensitivity while preserving reasonable development costs.

The requirements for this dissertation thesis have been coordinated with the results of an extensive survey of the state of art in this field. Our approach represents a method for context-sensitive automatic user interface generation. It relies on several methods introduced in the framework of this thesis. Our method for the user interface description and delivery supports consistent models on different levels of abstraction, as well as a seamless integration of the individual components of our system. As part of our approach, we suggest a context model that reduces development and maintenance complexity by the introduction of a concept of consistent independent sub-models. Our automatic user interface generation method uses optimization techniques to provide usable resulting user interfaces, minimizing the user effort required. The validity and efficiency of our approach are supported by an evaluation based on user studies, as well as an analysis of development efforts. Furthermore, this dissertation thesis describes a number of practical applications of our approach.

**Keywords:** Automatic User Interface Generation, Context Model, User Interface Description Language, User Interface Delivery, Model-Based User Interface Design, Optimization



# Abstrakt

Vývoj a široké rozšíření moderních informačních a komunikačních technologií představuje nové požadavky pro související uživatelská rozhraní. Je patrný nárůst diversity různých interaktivních zařízení jak co se týče jejich fyzické formy, tak i softwarové platformy, kterou tato zařízení využívají. Převážena většina dnešní společnosti vyžaduje přístup k moderním informačním technologiím a to včetně lidí s nejrůznějšími postiženími. Tento trend představuje výzvu pro související technologie pro uživatelská rozhraní. Tyto technologie by měly nabídnout dostatečnou míru adaptivity a přizpůsobení se kontextu použití při současném zachování rozumných nákladů na vývoj.

Požadavky na tuto dizertační práci byly koordinovány s rozsáhlou studií stávajících řešení v této oblasti. Navržené řešení reprezentuje přístup pro automatické generování uživatelských rozhraní zohledující kontext použití. Tento přístup závisí na několika metodách představených v rámci této dizertační práce. Naše metoda pro popis a doručení uživatelských rozhraní podporuje konzistentní modely na různých úrovních abstrakce stejně tak, jako konzistentní integraci jednotlivých komponent našeho systému. Jako součást našeho přístupu jsme navrhli kontextový model, který snižuje složitost vývoje a údržby prostřednictvím konceptu konzistentních nezávislých podmodelů. Naše metoda pro generování uživatelských rozhraní využívá optimalizačních technik tak, aby výsledná uživatelská rozhraní vyžadovala co nejmenší úsilí od svých uživatelů. Správnost a efektivita našeho přístupu je podpořena ověřením založeným na uživatelských studiích a na analýze úsilí nutného pro vývoj. Tato dizertační práce dále obsahuje několik praktických příkladů použití našeho přístupu.



# Acknowledgements

This research has been partially supported by the Technology Agency of the Czech Republic under the research program TE01020415 (V3C - Visual Computing Competence Center).

This research has been partially supported by Technology Agency of the Czech Republic, funded by grant no. TA01010784 (Form Cloud).

This research has been partially supported by the Czech Technical University under grant no. SGS13/213/OHK3/3T/13.

This research has been partially done within project Automatically generated user interfaces in nomadic applications founded by grant no. SGS10/290/OHK3/3T/13 (FIS 10-802900).

The research has been partially supported by project VITAL funded by the sixth Framework Program of European Union under grant FP6-030600 (VITAL).

This research has been partially supported by project i2home funded by the sixth Framework Program of European Union under grant FP6-033502 (i2home).





## Dedication

*To my parents.*



# Contents

List of abbreviations	xv
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Dissertation thesis objectives . . . . .	4
1.3 Structure of this thesis . . . . .	5
<b>2 Background and Related Work</b>	<b>7</b>
2.1 User Interface generation approaches . . . . .	8
2.2 Input for automatic User Interface (UI) Generation . . . . .	18
2.3 UI description languages and UI delivery . . . . .	22
2.4 Context model . . . . .	25
2.5 Automatic UI evaluation . . . . .	29
2.6 Conclusion . . . . .	32
<b>3 UI Description and Delivery</b>	<b>35</b>
3.1 UIP Platform . . . . .	36
3.2 UIP clients . . . . .	39
3.2.1 Common UIP Client Core . . . . .	40
3.2.1.1 UIP Client Event Protocol Communicator . . . . .	40
3.2.2 UIP Client Platform Extensions . . . . .	41
3.3 UIP Server . . . . .	42
3.3.1 Event Handling API . . . . .	43
3.3.2 UIP Application . . . . .	45
3.4 Conclusion and Contribution . . . . .	45
<b>4 Input for User Interface generation</b>	<b>47</b>
4.1 Abstract User Interface . . . . .	48
4.2 UIP visual editor . . . . .	50

4.3	Application audit and AUI transformation . . . . .	51
4.4	Transformation of specific input models . . . . .	53
4.4.1	Universal Remote Console Sockets . . . . .	53
4.4.2	OpenHAB . . . . .	55
4.5	Conclusion and Contribution . . . . .	56
<b>5</b>	<b>Context Model</b>	<b>57</b>
5.1	UIP Context Model . . . . .	57
5.1.1	Device Model . . . . .	58
5.1.2	User Model . . . . .	58
5.1.3	Environment Model . . . . .	59
5.1.4	Assistive Technology Model . . . . .	59
5.1.5	Computation of final context model property values . . . . .	59
5.2	Context sensors . . . . .	61
5.3	Conclusion and Contribution . . . . .	61
<b>6</b>	<b>User Interface Generation and Optimization</b>	<b>63</b>
6.1	Concrete User Interface (CUI) optimization . . . . .	63
6.1.1	Optimisation heuristics . . . . .	66
6.1.2	Templates . . . . .	68
6.2	CUI Generation Process . . . . .	68
6.3	Conclusion and Contribution . . . . .	71
<b>7</b>	<b>Source-code audit based UIs</b>	<b>73</b>
7.1	Application source code audit . . . . .	74
7.2	Resulting UIs . . . . .	75
7.3	Conclusions an Contribution . . . . .	77
<b>8</b>	<b>Application: Indoor navigation for users with limited navigation and orientation abilities</b>	<b>79</b>
8.1	Navigation system design . . . . .	81
8.1.1	Smart kiosk . . . . .	82
8.1.2	Interactive tactile map . . . . .	84
8.1.3	Simple navigation terminal . . . . .	84
8.2	SW and HW Architecture . . . . .	85
8.3	Conclusion and Contribution . . . . .	87

<b>9 Other applications</b>	<b>89</b>
9.1 E-governance . . . . .	90
9.2 GraFooSha: Food Sharing for senior users . . . . .	92
9.2.1 Technical realization . . . . .	93
9.3 Conclusion and Contribution . . . . .	95
<b>10 Evaluation</b>	<b>97</b>
10.1 Evaluation of context model adaptations . . . . .	97
10.2 Evaluation of perceived quality of generated UIs . . . . .	101
10.3 Development and Maintenance Efforts . . . . .	103
10.4 Evaluation of network transfer protocol . . . . .	104
10.5 Conclusion . . . . .	106
<b>11 Conclusions and Future Work</b>	<b>107</b>
11.1 Future Work . . . . .	113
<b>A Tools for development support</b>	<b>141</b>
A.1 UIP Web Portal . . . . .	142
A.2 UIP Visual Editor . . . . .	144
A.2.1 UIP Visual Editor Evaluation . . . . .	146
<b>B Source Code Examples</b>	<b>147</b>
B.1 UIP AUI Example . . . . .	147
B.2 Context model properties . . . . .	148
B.2.1 Device model . . . . .	148
B.2.2 User model . . . . .	149



<b>XAML</b>	Extensible Application Markup Language .....	23
<b>XML</b>	Extensible Markup Language .....	20
<b>UI</b>	User Interface .....	xi
<b>AUI</b>	Abstract User Interface .....	xvii
<b>CUI</b>	Concrete User Interface .....	xii
<b>UiGE Pipeline</b>	Context-sensitive Automatic Concrete User Interface Generation Pipeline .....	xvii
<b>FUI</b>	Final User Interface .....	12
<b>GUI</b>	Graphical User Interface .....	15
<b>CRF</b>	Cameleon Reference Framework .....	2
<b>UM</b>	User Model .....	15
<b>DM</b>	Device Model .....	23
<b>EM</b>	Environment Model .....	15
<b>ATM</b>	Assistive Technologies Model .....	57
<b>UCH</b>	Universal Control HUB .....	53
<b>CM</b>	Context Model .....	xix
<b>UiGE</b>	UIP User Interface Generator .....	xviii
<b>API</b>	Application Programming Interface .....	28
<b>RDF</b>	Resource Description Framework .....	26
<b>UIDL</b>	User Interface Description Language .....	3
<b>ORM</b>	Object-Relational Mapping .....	20
<b>MDD</b>	Model-Driven Development .....	9
<b>MBUID</b>	Model-Based User Interface Development .....	9
<b>PDA</b>	Personal Desktop Assistant	
<b>GP</b>	Generative Programming .....	16
<b>XSLT</b>	Extensible Stylesheet Language Transformations .....	19
<b>UML</b>	Unified Modeling Language .....	20
<b>DSL</b>	Domain-Specific Language .....	21
<b>AOP</b>	Aspect-Oriented Programming .....	20

<b>ICT</b>	Information and Communication Technologies .....	27
<b>OWL</b>	Web Ontology Language .....	26
<b>FP</b>	Framework Program .....	28
<b>UCD</b>	User Centered Design .....	32
<b>ISO</b>	International Standardization Organization .....	32



# List of Figures

1.1	Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline) . . . . .	2
2.1	Related work areas in relationship to the UiGE Pipeline . . . . .	7
2.2	Adaptability and adaptivity . . . . .	8
2.3	Architecture of MASTERMIND system . . . . .	10
2.4	Architecture of PUC system . . . . .	11
2.5	UIs generated by Uniform tool . . . . .	12
2.6	Functionally equivalent UIs generated by Supple . . . . .	13
3.1	UI delivery method in relation to UiGE Pipeline . . . . .	35
3.2	Client-server architecture with thick and thin clients . . . . .	36
3.3	UIP Protocol communication . . . . .	37
3.4	UIP Platform conceptual components and protocols . . . . .	37
3.5	UIP Protocol reference architecture . . . . .	38
3.6	Reference architecture of <i>UIP Client</i> . . . . .	39
3.7	Reference architecture of UIP server . . . . .	42
3.8	Simplified scheme of <i>Event Handling API</i> . . . . .	44
3.9	Components of a <i>UIP Application</i> . . . . .	45
4.1	Input transformation scope in the UiGE Pipeline . . . . .	47
4.2	Input Methods . . . . .	48
4.3	Example Abstract User Interface (AUI) structure . . . . .	48
4.4	Functionally equivalent CUI's . . . . .	49
4.5	Components of UIP visual editor . . . . .	50
4.6	Code-inspection . . . . .	52
4.7	URC integration – derivation of AUI . . . . .	54
4.8	Abstract Interface Builder for <i>URC</i> . . . . .	54
4.9	Integration of OpenHAB . . . . .	55

5.1	Scope of context model in the UiGE Pipeline . . . . .	57
5.2	Components of Context Model . . . . .	58
5.3	Computation of context-sensitive UI properties . . . . .	59
6.1	UIP User Interface Generator (UiGE) in the scope of the UiGE Pipeline	63
6.2	Simple mapping . . . . .	64
6.3	Example of simple mapping . . . . .	65
6.4	Heuristics rule . . . . .	66
6.5	Example of importance heuristic . . . . .	67
6.6	Template mapping . . . . .	68
6.7	Simplified scheme of the UI generation process . . . . .	69
6.8	Small model update . . . . .	70
6.9	Substantial model update . . . . .	70
6.10	AUI update . . . . .	70
6.11	Context model update . . . . .	71
7.1	Data-driven applications in the scope of the UiGE Pipeline . . . . .	73
7.2	UIs for iPad and PC generated with and without templates . . . . .	75
7.3	Adaptions of UIs for iPhone . . . . .	76
8.1	In-hospital navigation system components . . . . .	79
8.2	Navigation procedure with proposed system. . . . .	81
8.3	Early version of <i>Smart Kiosk</i> UI . . . . .	82
8.4	Example UI of <i>Smart kiosk</i> prototype . . . . .	83
8.5	Prototype if interactive tactile map . . . . .	84
8.6	Concept and physical realization of final prototype of Simple Navigation Terminal . . . . .	84
8.7	Client-server architecture of distributed navigation system . . . . .	85
8.8	Simple Navigation Terminal Components . . . . .	86
9.1	Applications described in this chapter in the scope of the UiGE Pipeline	89
9.2	Transformation from 602 zfo to <i>UIP</i> CUI and AUI . . . . .	90
9.3	Form rendered using original Form Filler Software . . . . .	90
9.4	Same form rendered using UIP Desktop Client . . . . .	91
9.5	Equivalent form as a result of automatic CUI generation . . . . .	91
9.6	Design developmnet of GraFooSha . . . . .	93
9.7	Internal Electronic Components of GraFooSha Device . . . . .	94
9.8	Integration of GraFooSha IoT device into the <i>UIP Platform</i> . . . . .	95

10.1	Plan of the three-phase Context Model (CM) evaluation study. . . . .	98
10.2	Example of user interface used in the study . . . . .	99
A.1	Scope of the <i>UIP Development Support Tools</i> in the framework of <i>UiGE Pipeline</i> . . . . .	141
A.2	<i>UIP Portal</i> – <i>UIP Application management</i> . . . . .	143
A.3	Low-Fidelity prototype of <i>UIP Visual Editor</i> . . . . .	144
A.4	Evolution of property window of the Low-Fidelity prototype of the <i>UIP Visual Editor</i> . . . . .	145
A.5	<i>UIP Visual Editor</i> – editing <i>AUI</i> . . . . .	145
A.6	Deployment of a <i>UIP Application</i> from the <i>UIP Visual Editor</i> . . . . .	146



# List of Tables

2.1	Comparison UI generation approaches . . . . .	16
2.2	Comparison of input methods suitable for AUI derivation . . . . .	21
2.3	Comparison of UI description languages . . . . .	24
2.4	Comparison approaches for context modeling . . . . .	28
2.5	Comparison of UI evaluation methods . . . . .	31
8.1	Context model properties relevant for CUI adaption . . . . .	83
10.1	Results of CM evaluation study . . . . .	100
10.2	Subjective assessment of context model properties . . . . .	101
10.3	Subjective evaluation of form UIs . . . . .	102
10.4	Comparison of manual and code-inspected approach regards the size of code. . . . .	103
10.5	Frame transfer time (Simulated stream) . . . . .	105
10.6	Frame transfer time (Real-time video) . . . . .	105
B.1	Device model properties . . . . .	149
B.2	User model properties . . . . .	151



# Chapter 1

## Introduction

The rapid development of information and communication technologies in recent decades has brought about new requirements for interactive systems. Modern applications should be able to run on devices of various kinds, capabilities or based on different operating systems. Moreover, UIs of such applications should adapt to the current context of use. The aspects of the current device on which the application is running should be considered. It is also very important to adapt to the needs and preferences of the current user, as well as to the current environment in which the interaction carried on. In some cases, one application session can migrate over multiple devices during the runtime; some application UIs are also distributed across multiple devices to enable more effective and more comfortable interaction, e.g. for purposes of smart household control. Details regarding new requirements brought about by this development are described in section 1.1.

When dealing with the abovementioned requirements, standard UI development methods can be very inefficient. According to [16], in such cases, massive code replication that complicates both application development and maintenance is the most serious issue. This thesis deals with context-adaptive automatic UI generation for heterogeneous UI platforms. It introduces *UIP* (User Interface Platform) that simplifies UI development using multiple novel concepts. We introduce a method of how to deliver a single *UI* description to multiple device platforms that differ in software and hardware capabilities, as well as in the operating system. The current rise of popularity of interactive devices used in various contextual situations requires novel methods to deal with new *UI* adaption requirements. The primary contribution of this thesis is the introduction of a method for automatic *UI* generation. On the basis of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline), this method enables automatic context-sensitive trans-

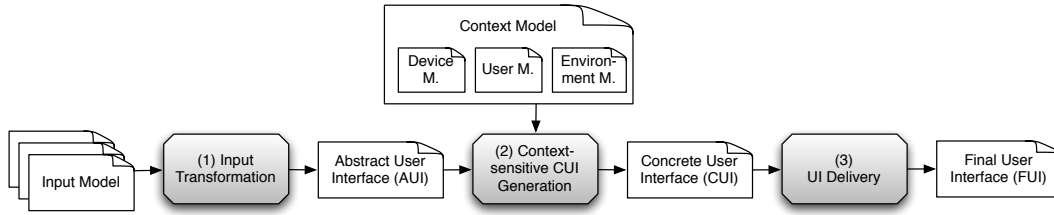


Figure 1.1: Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline), inspired by [11]

formation from one input model to multiple concrete UIs whose appearance and structure corresponds to the actual contextual situation. A context-model that fits to proposed UI generation methods is also introduced as part of this thesis.

This dissertation thesis focuses on the context-sensitive automatic generation of UIs. Approaches in this research domain typically reference the Cameleon Reference Framework (CRF) [11] as a guideline for specification of basic definitions and terms. CRF differentiates the evolution of a UI into four stages:

- *Concepts and Task Model* are the most abstract stages. Various models that describe UIs on this level exist, including *Task Model*, *Workflow Model* or *Dialog Model*.
- *Abstract User Interface (AUI)* is a hierarchical composite structure that describes UIs in a platform-independent manner. AUI does not define actual element representation or the layout.
- *Concrete User Interface (CUI)* is an explicit description of UI. CUI description consists of specific UI elements (interactors). Although actual CUI form depends on the target platform, the CUI description language is platform independent.
- *Final User Interface (FUI)* is UI in a form that can be rendered on a particular run-time infrastructure platform (e.g. iOS Smartphone, Windows PC). It uses platform native UI elements (interactors).

This general definition is also represented in Figure 1.1. Individual approaches for UI generation differ from others in the principles behind individual steps of the generation process and their complexity. For the purposes of this thesis, the process of automatic generation of CUIs is defined as context-sensitive transformation of an AUI into a CUI. Context-sensitivity includes adaptations to target platform, user and



interaction environment. The resulting UIs are optimized using minimal metrics, primarily to minimize the estimated user effort necessary for the interaction.

This thesis focuses on the realization of the CUI generation pipeline as depicted in Figure 1.1. Individual steps of this pipeline were addressed by different methods described in particular chapters of this dissertation thesis. We call the overall realization *UIP Platform*. From the perspective of the UiGE Pipeline, this dissertation thesis focuses on all its steps. In chapter 4, we describe a method of how to derive input from various models used in the industry. Chapter 5 describes the Context Model (CM), which is an important attribute of the process of automatic CUI generation. Our method for automatic CUI generation is described in chapter 6. Chapter 3 focuses on our User Interface Description Language (UIDL) and our method of CUI delivery, as well as on defining the basic foundations of the *UIP Platform*.

Apart from a description of automatic CUI generation pipeline realization on the theoretical level, this dissertation thesis also focuses on practical implementation. Chapters 7 - 9 focus on practical applications in which the methods described in this thesis were used. From a developer perspective, the more steps of the UiGE Pipeline that are used for an implementation of an application, the less effort is expected to be invested into its development. In appendix A we describe the development support tools that have been created in relation to our realization of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline). Namely, we mention related *UIP Visual Editor* (see section 4.2 and section A.2 and development support web portal (see section A.1).

The following section describes in detail the motivation for the approach and corresponding methods introduced in the framework of this thesis.

## 1.1 Motivation

Users often perceive the quality of the UI as the quality of the whole application. The development of application UIs with a conventional approach typically requires significant implementation efforts. According to [48], about one half of an application code is related to its UI. The application development costs are therefore significantly affected by the effort invested into the application UIs [wos4]. When considering adaptive UIs, the costs and effort grow with the number of features that are provided and the range of user groups that are supported.

The development of UIs is currently reaching the post-WIMP era [109]. This

trend is most obvious in the rise of smart devices, especially smartphones and tablet computers. Recently, analysts have reported that the number of smartphone users worldwide has reached one billion [14]. This implies a need to focus the UI development on new UI platforms. Smart devices have various form-factors and hardware specifications, and are based on various operating systems. In most cases, it is necessary to develop an application that runs natively (using platform native UI components) on multiple platforms. This requirement typically leads to code-replication when the platform-dependent UI part of the application must be restated for each supported platform.

From a development perspective, it is challenging to deal with multiple platforms. Each platform typically has more or less different development mechanisms and underlying programming languages too. If there is a demand for adaptive UIs that reflects changes in the current usage context, the problem of UI development becomes even more complex. In such a situation, it would be very difficult to manually implement UIs that satisfy the requirements of all the possible contextual situations. There would be a large amount of restated UI code for individual situations. Consequently, the development and maintenance costs for the UIs would be very high. A possible solution to this problem is to address it through automatic UI generation.

## 1.2 Dissertation thesis objectives

This section summarizes the general objectives of this dissertation thesis. The primary aim, as well as the title of this thesis, is to model an automatic UI generation pipeline. Accomplishing this aim requires addressing the challenges stated in the previous section 1.1. A list of objectives that frame the aim of this dissertation thesis follows:

1. Definition of a methodology for an *Automatic Context-sensitive Generation of Concrete User Interfaces*.
2. Modeling and Implementation of this methodology in a form of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline).
3. Integration or Development of a *Context Model (CM)*. An existing suitable *Context Modeling* method could be adapted. Alternatively a novel context-modeling method that suits our requirements regarding the development ef-

efficiency and consistency with other UiGE Pipeline components can be developed.

4. Integration or Development of a *UI Description and Delivery Language/Method*. It should be an integral part of the UiGE Pipeline implementation and support high development efficiency.
5. Minimization of the amount of work required for UI development and maintenance in case of complex multi-platform, context-sensitive UIs.
  - (a) Provide developers with supporting tools and usage guidelines that will help them to deal with the UiGE Pipeline.
  - (b) Development of an *input transformation* method for input derivation from existing models used in practice.
6. Evaluation of the proposed solution.
  - (a) Demonstration of its usefulness for purposes of practical applications.
  - (b) Evaluation of generated UIs from the perspective of relevant target users.
  - (c) Evaluation of related development support tools from the perspective of their users (the developer perspective).

## 1.3 Structure of this thesis

This thesis is structured as follows: chapter 2 focuses on related approaches and methods relevant to the objectives of this thesis. It lists comparable UI generation approaches, possible input models, UI description languages sufficient for automatic UI generation, context modeling methods, and methods of automatic evaluation of UI quality.

The following chapters focus on the main topic of this thesis – design and implementation of a platform for automatic UI generation. Chapter 3 focuses on the software architecture of the platform and primarily focuses on methods to deliver a single UI to different platforms that differ in capabilities. Chapter 4 describes different methods that are considered to comprise an AUI – the immediate input for automatic UI generation is defined in the terms of this thesis. Chapter 5 describes our multi-component context model which was defined for purposes of this work.

Chapter 6 focuses on the automatic CUI generation method as the main objective of this thesis. Mainly, it describes the UI optimization function, as well as the realization of the CUI generation process itself. After the definition of methods that represent a contribution of this thesis, we list selected applications that were based on those methods. In chapter 7, we show an example that uses application source-code audit to generate context-sensitive CUIs. An indoor navigation system tailored for navigation of individuals with low navigation and orientation abilities is described in chapter 8. Finally, chapter 9 lists other substantial real-world applications that at least partially incorporate methods brought about by this work.

The evaluation of individual concepts introduced by this thesis, as well as their applications, is described in chapter 10. After an evaluation of UI quality perceived by target users, we also focus on the analysis of the development effort to investigate the efficiency of our method. The evaluation of important applications of our approach is also part of this chapter. This thesis is concluded by chapter 11, which describes the level of fulfillment of the individual thesis objectives. A statement of possible future work is also part of this final chapter. Appendix A focuses on development tools related to the UiGE Pipeline. The second appendix, B, contains technical examples related to our UI description language and Context Model.

# Chapter 2

## Background and Related Work

This chapter focuses on analysis of related research relevant to the automatic UI generation. At the beginning, general approaches for automatic UI generation are listed in section 2.1. Figure 2.1 shows to which parts of the UiGE Pipeline refer the individual sections of this chapter. Discussion of several methods and approaches to derive immediate input for automatic CUI generation is described in section 2.2. One of the important objectives of this work is to generate UIs that can be delivered to various platforms with different capabilities and operating systems. In section 2.3 we focus on relevant User Interface Description Languages that are suitable for platform-invariant UI description.

As the context-model is an important attribute to the automatic UI generation, different methods for context-modeling are stated in section 2.4. Quality of UIs is typically evaluated by methods dependent on human experts like usability stud-

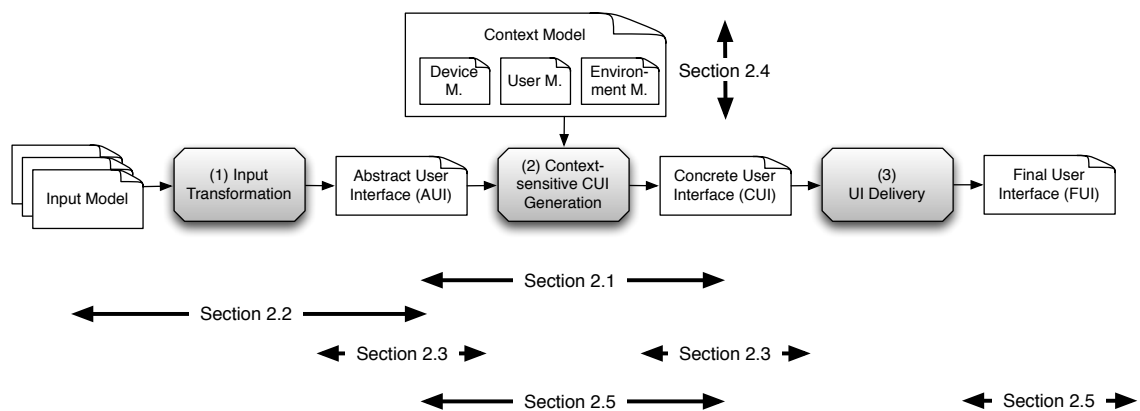


Figure 2.1: Related work areas in relationship to the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

ies or expert evaluation methods. In case of automatic UI generation, automatic methods must be used to assess quality of resulting UIs in order to provide optimal results. Possible approaches for automatic UI evaluation and optimization are listed in section 2.5.

## 2.1 User Interface generation approaches

Basic approaches to simplify UI design are visual editors and widget builders, such as [70]. While these tools help with the initial design for a particular platform, they provide only limited support for maintenance. For example, when a developer wants to generalize or to parametrize the UI code, this type of builder fails to adapt to the code changes, so that subsequent changes cannot be made through the editor [48].

There are approaches like Microsoft Access [21] and Oracle Forms [91] that use widget builders to compose form-based UIs to access data in relational databases. These approaches also support the semi-automatic generation of the forms. Although these systems are well adopted by the industry due to their simplicity, they are not suitable for generating context-sensitive UIs for multiple platforms. They lack context model adaptations, there is no support for various layouts and custom components and the output platform is restricted to desktop and web.

The fundamental work on model-based development has been conducted by Stephanidis et al. [104]. In their work they provide an overview of adaption techniques in the web environment. They distinguish between *adaptability* and *adaptivity* terms, see Figure 2.2. *Adaptability* refers to self adaptation based on knowledge available prior to interactive sessions, while *adaptivity* refers to self-adaptations based on knowledge acquired during the runtime. Corresponding project *Avanti* is presented

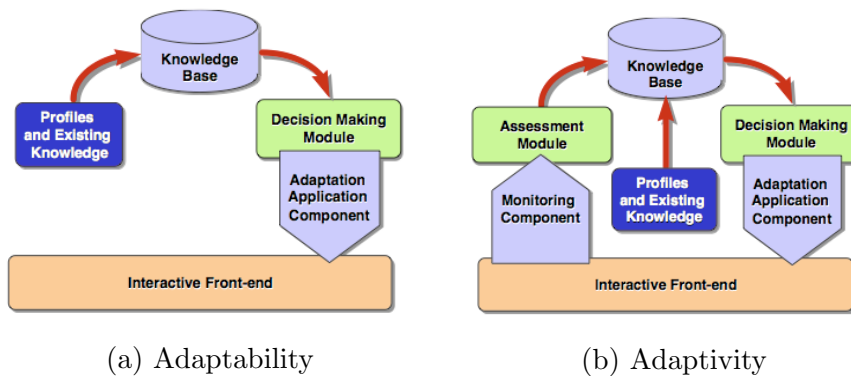


Figure 2.2: Adaptability and adaptivity, from [104]

as a case study to demonstrate adaptivity features. Among other properties, the presented system adapts to context properties like disabilities, expertise or interests of the user, or even to the season of the year. The context knowledge is obtained by questionnaires (disabilities, expertise, interests) or from system resources (season). This context model structure corresponds to the examples shown in the framework of the *Avanti* demonstrator, however, it might be inappropriate for another practical applications.

Wobbrock et al. introduced concept of ability-based design [113]. Similarly to Stephanidis et al. [104] they distinguish between *adaptive* and *adaptable* systems. According to Wobbrock et al., *adaptivity* mean the degree to which a system can change itself in response to user behavior, whilst *adaptability* mean the degree to which software can be customized by a user, therapist etc. This definition is consistent with that by Stephanidis et al. The consequence of this definition is that adaptivity requires an automatic UI generation that reflect knowledge (typically in form of Context Model) in case of nontrivial cases.

Sottet et al. [101, 102] focus deeply on Model-Driven Development (MDD). In their work they provide a deep explanation of MDD approaches to model-code and model-model transformations. They describe an approach for (semi)automatic generation of ubiquitous UIs preserving usability. They defined transformation mappings that preserve usability properties. Authors state that the ergonomic and usability criteria defined by mappings are often inconsistent and the final solution may require trade-offs. They provide an example of a home heater system control showing a framework for usable UI. Their work has some limitations, for example element composition does not allow simple modification or parametrization of a specific UI element. For instance, a single element cannot be easily positioned relative to others. From the practical point of view the presented framework lacks compatibility with traditional development approaches such as JavaEE [8]. This can limit its practical usage, impact the performance, and in combination with an existing application backend, this approach leads to information replication in the model.

Calvary et al. [11] introduce a unifying reference platform for developing multi-context UIs. The context of use is decomposed into user, platform and environment context. The authors also introduce the notion of plastic UIs, which support multiple contexts of use while preserving usability as context-adaption occurs. The term *plasticity* is also mentioned by Sottet et al. in [101, 102]. Several Model-Based User Interface Development (MBUID)-based approaches are evaluated against the proposed reference platform. For such systems, integration with the application backend

will introduce possible code-restatement. This makes development and maintenance more difficult. Context models in the reference framework are ontology-based. However, the expressive power of ontology-based models is strong, corresponding development and maintenance are very complex for real situations. This makes it harder to use such systems in a real environment.

In complex systems, multiple different interactive devices can render the UI. These devices are often based on various platforms and have various capabilities, e.g. resolution, size, interaction modality, etc. Technologies that allow a single UI to be delivered to various platforms are already available, such as HTML 5, but constraints on adaptive features and context-awareness persist. Although there are approaches that suggest partial-solutions, such as [10], they do not provide a general solution and typically fail to provide real-time context-aware adaptations and are limited to a small number of context properties.

MasterMind [105] is one of the first systems that integrate multiple models together. As shown in Figure 2.3, an application model, task model and presentation model are used in the design-time environment. Proprietary notations for all models were used in the MasterMind system. This makes this system rather good example of early model-based toll than reusable approach.

The Mobi-D system described in [87] provides assistance in the development process rather than automatic design. Interface and application developers are still involved in the development process. The Mobi-D system is a set of tools to support development cycle. There are several clearly defined models: user-task model, domain model, dialog model and presentation model. Relations between these models are also explicitly defined. The development process starts by deriving user tasks,

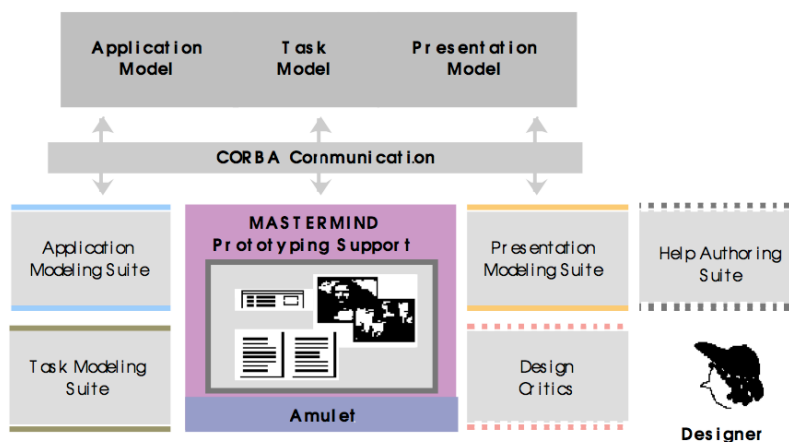


Figure 2.3: Architecture of MASTERMIND system (from [105])



starting by informal description converted to an outline. The next step is the definition of user-tasks and the domain. Skeleton domain model is derived from task outline. Both domain and user model are refined by a developer. The framework provides explicit methods for generalizing pieces of models to be reused in other designs. The final step is design of presentation (user interface) and dialog. Decision support tools provide recommendations in order to help developer to build the final interface. This system provides recommendations that do not limit flexibility, but make the development process more organized. The Mobi-D system is supposed to be used by usability engineers rather than by standard developers. The models for more complex interactions require significant effort to be developed. For purposes of this work is important an idea of explicit separation of models. Furthermore reusability of pieces of models can lower required development effort of any system and provide more consistent results.

XWeb system described in [77] tries to apply the web metaphor to services in general in order to support higher levels of interactivity. The main motivation is to enable creators of services to abstract from interactive technologies and devices. Although neither XWeb was massively practically used, it brings an important idea of platform-based thin client. This is actually generalized web approach that becomes more and more adopted by Web 2.0 technologies like Google docs. The idea of moving the application logic to the server and supply the user with as thin platform-specific client as possible inspired the development of our solution for CUI delivery – the *UIP Protocol*, see chapter 3.

The main motivation for the Personal Universal Controller (PUC) [71] project was to simplify controlling of appliances by using a single device with richer UI that

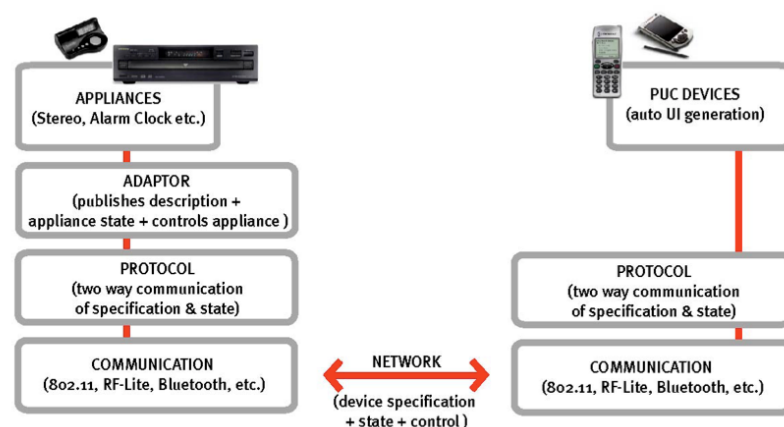


Figure 2.4: Architecture of Personal Universal Controller system (from [71])

is able to control more appliances at once. Mobile devices like PDAs or Smart phones were used as the main controllers. Figure 2.4 depicts the architecture of the PUC showing one connection between a PUC device and an appliance. This complex system allows controlling various appliances by various controlling devices (PUC devices). The PUC use its own proprietary appliance-oriented language for describing abstract user interfaces. For generation of CUIs, the PUC uses a simple rule-based approach (e.g. a command is always represented by a button). An advantage of the PUC project is that it is also to generate speech UIs.

Uniform [72] is another promising user interface generator that has been brought into public in 2006. The main contribution is that the consistency of the Final User Interface (FUI) is taken into account. It means that the system is trying to find similarities between currently generated user interfaces and interfaces that have been presented to the user in the past. The final look of the user interface is therefore adapted to be as consistent with current user interfaces as possible. Figure 2.5 shows an example of user interfaces generated by Uniform. The first two images show the user interfaces of two independent copy machines. On the second two images there are depicted UIs rendered in order to be consistent with copy machine A or B respectively.

ICrafter [85] is a framework for services and their user interfaces in a class of ubiquitous computing environments. Authors refers the ubiquitous computing environment as an interactive workspace. The aim is to enable users to interact with services in this environment using various modalities and input devices. ICrafter provides UI generation and on-the-fly aggregation of services. Accordingly to the



Figure 2.5: User interfaces generated by Uniform tool for a simple and complex copy machine with consistency attribute disabled and enabled (from [72])

authors, a service is either a device (e.g. light, projector or scanner) or an application (e.g. web browser or PowerPoint running somewhere in the environment). In Appliances request UI from interface manager, the request contains the appliance description. At first, the interface manager selects an appropriate UI generator (a software entity that can generate UI for one or more services for a particular appliance). The appropriate generator is selected in the following order: generator for the service instance, generator for the service interface and finally service independent generator as a fallback. In the next step the selected generator is executed with access to service descriptions, appliance description and the context. Using this information, the appropriate generator constructs the FUI.

Unlike other systems, ICrafter uses specific *UI generators* for particular services and UI description languages (target platforms). Most of them are implemented using a template system. An interesting idea brought by the ICrafter approach is the usage of so-called service patterns. Patterns are recognized in the available services. ICrafter then generates UIs for these patterns. On one hand, this leads to better consistency and easier service aggregation. On the other hand, unique functionality is not available in the aggregated service. Another contribution is the involvement of a template system (parametrizable, human-designed parts of a UI) in UI generators. In most cases, the UI designed by human designer is better than automatically generated equivalent (exceptions are for example UIs generated fit complex contextual restrictions, e.g. UIs for people with specific needs).

Supple [37] uses combinatoric optimization to generate CUIs optimized according

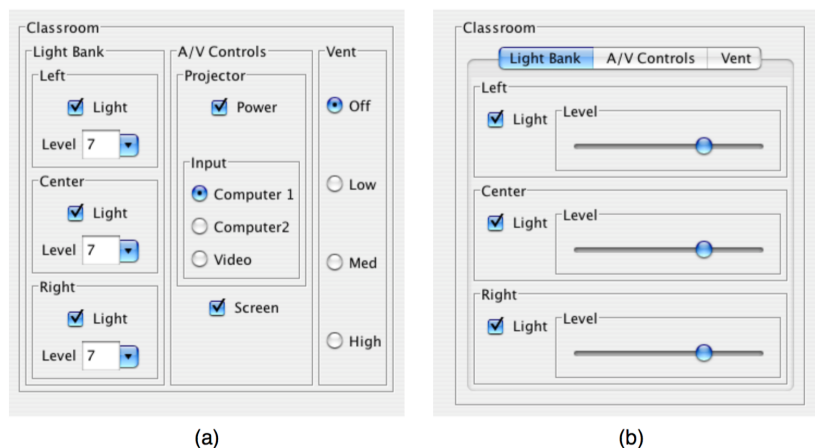


Figure 2.6: Functionally equivalent UIs generated by Supple. The difference in UIs is caused by parametrization of optimization function. Where for UI (a) is preferred easy navigation, whereas for UI (b) are preferred easy to use widgets. (from [37])

to minimal metrics. Combinatoric optimization is based on a functional description of a UI, and takes into account both device and user properties. The UI generation process is defined as an optimization problem where the algorithm is trying to minimize the estimated overall user effort. Figure 2.6 shows example UIs generated by Supple. These functionally equivalent UI are generated with the same space constrains, but using different parametrization of the optimization function. User behavior is traced in order to adapt the generation of the new UI to user properties and user needs (recognized from prior use or using special motor and cognitive tests).

Approach by Jelinek et al., described in [46] is an example of an approach that uses an annotated source code as input. The motivation was to remove the necessity to manipulate with abstract models explicitly, but keep the flexibility they provide. The authors define the AUI as a set of the following elements: text input, number input, single item selection, multiple item selection, monitoring, responding to alerts and other (specific) elements. A simple approach for generation of concrete user interfaces was used – an explicit platform dependent mapping of concrete widgets to abstract user interface elements. Furthermore, a simple vertical layout was used. This system is designed to support ubiquitous computing (in this case an interactive workspace). There is a key idea of services (devices or applications) that are used as an input for the process of user interface generation. Generation of the concrete user interface depends on the selected generator. Usually the generator is platform and service specific, but a simple automatic approach is possible.

In [32], Engel et al. provide an evaluation of model-based UI development approaches. Furthermore, they present their own PaMGIS framework that supports semi-automatic generation of UI code. This MBUID approach based on various input models, namely: task, dialog, interaction, layout, user, device, and environment models. Complexity of the models is reduced using patterns on different abstraction levels. The authors also introduce their own PaMGIS pattern specification language (PPSL).

Vanderdonckt in [110] focuses on distributed user interfaces (DUIs). In contrast to traditional UIs, DUIs enables users to distribute UI elements among different platforms and physical environments. This work provides a conceptual framework to support UI distribution controlled by the end user, under control of the system or a mixed-approach. Resulting UIs can be subject to adaptation with respect to the end user, environment and the target platform.

Kolb et al. in [52] focuses on automatic generation of UI component in process-

aware information systems (PAISs). The PAISs focusses on business processes and are mainly used in large companies. The authors state that there was little effort to automatically generate UIs in this domain, despite the process of manual UI creation is tedious and error-prone. According to the authors, UI logic can be expressed using the same notation as for process modeling. They introduce a bi-directional pattern-based transformation between process model and the UI.

Tran et al. in [106] present an approach for multi-agent system (MAS) based UI generation from combined task, context and domain models. In this scope an agent is *a computer system, situated in some environment that is capable of flexible autonomous action in order to meet its design objective*. The described system uses four types of agents: Model analyst, Function analyst, UI creator, and Code generator. These agents gradually transform the input models into UI code and application logic code. The authors present their approach on an example of basic database application.

Cerny et al. in [16] present a system for automatic UI generation accordingly to user's specific preferences. Resulting UIs conform to user different skill levels, capabilities or physical locations. While reducing development and maintenance costs, the authors suggest to application source-code audit and Aspect-oriented programming for UI development. UI adaptations suggested by the authors are rather on level of UI structure. The proposed approach should be extended to support more target UI platforms as well as adaptations on the level of UI appearance, e.g. the widget selection.

UI generation approach	Principle	Context-sensitive	Pros	Cons
Widget builders – e.g. NetBeans [70]	Manual Graphical User Interface (GUI) design	No	Universal approach, easy to use	Little adaptation (mostly OS level), code replication
Widget builders for databases – e.g. Microsoft Access [21] and Oracle Forms [91]	Manual and semi-automatic GUI design	No	Easy to use	Little adaptation, limited to relationship databases
Stephanidis et al. [104]	MDD	Yes (User Model (UM), Environment Model (EM))	Adaptivity	Complicated development and maintenance of model descriptions

Sottet et al. [101, 102]	MDD	Yes	Ubiquitous UIs, usability optimisation	Incompatible with traditional development methods, information replication
MasterMind [105]	MDD	No	Early MDD tool	Proprietary notations, obsolete
Mobi-D [87, 86]	MDD	Yes	Early complex MDD approach	Obsolete, rather development assistance tool, high development effort, only explicit mapping
XWeb [77]	Tree remapping, static rules	No (only attributes of target device and platform)	Idea of platform-specific thin client	Static transformation rules
Personal Universal Controller [71]	Rule based transformation	No	Multimodal, multidevice	Proprietary description languages, obsolete
Uniform[72]	Rule based transformation, consistency rules	No	Optimises UI consistency	Some principles restricted to home appliances domain
ICrafter [85]	Template based transformation, service aggregation	Yes (environment)	Multimodal ubiquitous approach, UI patterns	Obsolete, platform specific templates must be developed for each service/appliance
Supple [37]	Combinatoric optimization	Yes	Global combinatoric UI optimisation	Complicated support of multiple target platforms
Jelinek et al. [46]	Generative Programming (GP)	No	Development efficiency	Simple platform-specific transformation rules only
PaMGIS [32]	MDD	Yes (user, device, environment)	Complexity reduction by abstraction patterns	Complex maintenance of many different models
Vanderdonckt in [110]	Conceptual framework	Yes (user, device, environment)	UI distribution	Not fully implemented
Kolb et al. [52] (Process model transformation)	PAISs, business rules	No	Bi-directional transformation between process model and UI	No adaptivity, UI optimisation
Tran et al. [106] (Agent Based)	Multi-agent system, model transformation	Yes	Easy to use	Database example only
Cerny et al. [16]	AOP, code inspection	External	Effective development and maintenance	No global optimisation

Table 2.1: Comparison UI generation approaches

In this section, we have listed different approaches that deal with the automatic UI generation. In the Table 2.1 there is a summary of those related approaches accordingly to their main principles, their pros and cons. Several listed methods have brought interesting, original concepts, however, they are now obsolete. For example, the Personal Universal Controller [71] focuses on a platform that is not practically used anymore.

The MasterMind project [105] uses proprietary notations, Mobi-D [87, 86] is rather a development support tool than a true automatic UI generation method. ICrafter [85] brings interesting ideas of service integration based on their hierarchy and transformation templates, however, the realization from 2001 is also technically obsolete. Some approaches also focus only on a specific domain, e.g. Uniform [72] focuses the home appliance control only.

Generally, we can focus on the advantages and disadvantages of a particular method from the user perspective, but also from the perspective of application developers. In accordance with this thesis objectives, an ideal approach should provide maximally usable UIs that are adapted to the user needs and preferences. At the same time, it should support an effective development with low information restatement. Some listed methods are conceptually strong, but require high development and maintenance costs for a real deployment. E.g. PaMGIS [32] bases on many different models that must be maintained and also it does not support minimization of information restatement. MDD approach by Sottet et al. [101, 102] is incompatible with traditional development tools and also induces unnecessary information restatement. Stephanidis et al. [104] described one of first MDD approaches for automatic UI generation. This work also contains important definitions, however related context model structure is complicated and corresponding examples are obsolete.

However, Supple has a limited support of target UI platforms, it brings an important concept of combinatoric optimization – minimization of estimated user’s effort. This advanced CUI optimization method inspired also our method described in chapter 6. In contrast, some approaches support only simple AUI to CUI transformation based on static mapping rules, e.g. [77, 46]. This simple transformation can not provide optimal results in complex cases. For purposes of our work, we used more sophisticated transformation inspired by the combinatoric optimization.

Current trends as stated in the motivation section (1.1) require a solution that

is able to effectively generate UIs that are accustomed to their context of use. Furthermore, a satisfactory method should enable effective support for multiple target UI platforms at the same time. Another requirement emerges from the perspective of development and maintenance costs. AspectFaces [16] for example bring an interesting solution, primarily from the perspective of development efforts. However, in this approach context adaptivity is restricted to the general UI structure rather than on adaption on the presentation level (i.e. widget selection).

The approach described in this thesis should focus on maximizing the usability of resulting UIs and the versatility in terms of the number of supported platforms. It should enable complex UI adaptations of the presentation level and context sensitivity. At the same time, it is necessary to focus on minimizing required development and maintenance efforts.

## 2.2 Input for automatic UI Generation

In the previous section, we focused on a general survey of approaches for automatic UI generation. These approaches use various input models with different complexity. This section focuses on input models suitable for automatic UI generation. Consequently, we focus on related methods to effectively derive AUIs from other input models, ideally those already used in the field.

From the perspective of input for automatic UI generation, various approaches have been developed. These approaches can be divided into generation approaches (GP), model-based approaches (MDD), inspection-based approaches, and aspect-based approaches. Each of these offers certain advantages for UI development, but may fail to address UI maintenance or complex situations, mostly when the UI changes during runtime or adapts to users. In this section we focus on approaches that enable generating AUI as an output. Although AUIs can be implemented manually, this would be complicated for complex applications. In many cases, it is better to derive the AUI from other available input model or even using analysis of an existing application source code.

Model-driven development (MDD) [18, 83] suggests that a model is the source of information and the resulting code is generated using this model and a set of transformation rules. Model-Based User Interface Development (MBUID) [11] is a variant of MDD in the domain of UI development. The main advantage should be that there is no information replication, but this applies only to basic scenarios.

MBUID model transformations are addressed by Clerckx et al. [20]. In more



complex cases, inconsistencies between the source and derived models become an issue. When modifying concrete models like a dialog model or CUI, developers introduce inconsistencies that should be also reflected in source abstract models. A solution that partly solves this problem is bi-directional mapping, which ensures that most modifications in derived models are automatically reflected in the source models. The proposed solution was demonstrated on the *DynaMO-AID* prototyping tool.

Based on Lopez et al. [59], most current adaptive systems use hardcoded adaptation rules that are hard to maintain and reuse. Code replication may become an issue when capturing multiple variants of usage context. The authors suggest to use MBUID together with transformations from abstract models to concrete models based on generic mapping rules. They present the *T:XML* tool for a graphical specification of the transformation rules. As output it provides adaptations for transformation languages such as Extensible Stylesheet Language Transformations (XSLT). The target language to describe UIs in a platform independent manner is *USIXML* [58]. It involves multiple abstractions to capture the specification of UI in multiple context variants while supporting platform independence. The authors conclude that excessively detailed adaptation rules are hard to reuse in other applications and thus they suggest providing a set of generic mapping rules.

A possible advantage of MBUID approaches based on task models [7] or workflow models [98] is that an interaction flow (UI navigation) can be automatically derived. According to [63], in practice, most applications lack such model definitions due to their complexity. To avoid the unnecessary code-replication, state transitions based on the workflow description should be tightly integrated with an application low-level source code [98]. Typical data-oriented applications with form-based UIs include an implicit interaction flow that results into standard navigation (e.g. validation phase, possible decomposition of complex forms into a wizard UI etc.).

Luyten [60] aims to apply MDD based on a task-centered approach to fill the gap between HCI design and software engineering. The authors suggest to use the Concur Task Tree (CTT) notation [7] to model tasks in environment context-aware manner. The focus of the work is on distributed UIs. A tool *MoDIE* is presented for CTT modeling in accordance with an environment model. Similarly to [101], this approach does not suggest a concrete connection to the existing applications or application backends.

A workflow model or a task-model, such as [7], can be used to derive AUI. In this case, the task-model must be very detailed in order to enable individual UI

components to be derived, including restrictions (used for final element selection, validation etc.), data-binding etc. Development and maintenance of such models for complex applications is demanding. Alternative approaches such as Generative programming [24] use domain specific languages (DSLs) to address the separation of concerns, but they lack runtime context awareness and can introduce information replication. An example of such an approach is the method described in [93].

Aspect-Oriented Programming (AOP) is efficient for the transformation of primary input models such as a data-model to the *AUI*. Aspect-oriented approach [55] seems to be an interesting approach that brings the runtime weaving process, although still does not address the problem of information replication.

An approach that addresses the above mentioned disadvantages involves the use of data persistence structures code-inspection, AOP and GP concepts. Modern object-oriented applications use Object-Relational Mapping (ORM) [78] to bridge the database incompatibility [8]. Code-inspection applied to the extended data model can be used for deriving AUIs [18]. The data model extensions are similar to the Unified Modeling Language (UML) stereotype extensions [17]. In source code is this extension achieved by field annotations or by Extensible Markup Language (XML) descriptors. This type of extension is not novel – the industry already uses these annotations for input validation and for security [15].

Data model extensions are not limited to the concerns mentioned above. Such an extension can consider user profiles, location-awareness, etc. [18]. The result of the application data inspection is a hierarchical structural model that can be transformed to AUIs in multiple steps. An aspect-oriented approach can be used for the purposes of this transformation. This ensures that there is a small amount of replicated code and easier maintenance. This process takes place at runtime, it reflects the application data-model and the current context, and allows the system to adapt to various situations. The advantages over the MDD approach are that it does not require the presence of an extra model for the UI and thus it does not restate any information in it.

Table 2.2 summaries input methods suitable for derivation of AUIs as immediate input for CUI generation method as subject of this thesis. Authors of [67] observe that MDD induces problems during adaptation and evolution management. MDD handles common situations well, but when we need a slight modification of UI for an edge case this would take place in the UI code rather than in the model itself [16]. Potential code regeneration from the model becomes impractical, as it erases the above mentioned manually added information [16]. Another issue arises when we

Table 2.2: Comparison of input methods suitable for AUI derivation

Input method	Examples	Pros	Cons
MBUID	[83, 11, 20, 59]	Universal approach	Often require information replication
GP	[93]	Addresses concern separation	Problematic runtime adaption, possible information replication
Task and Workflow models	[7, 98]	Universal approach	Complex development and maintenance of detailed models
Code inspection + AOP	[48, 18, 16]	Effective, low code replication	For applications using ORM (certain inspection method is domain-specific)

apply the MDD approach solely to the UI, but not to the persistence and business part of the system. In such a case, the information captured by the model must match to the information captured by the rest of the system. When only one part of the system changes, another part may lose compatibility and may need to address the changes. When Domain-Specific Language (DSL) are used for the UI description, these languages do not provide type safety and thus maintenance of change propagation becomes tedious and error-prone since changes are made manually.

AspectFaces [15] and MetaWidget [48] are examples of code-inspection approaches with an ability to inspect Java-based data models and to understand their extension marks. MetaWidget, however, does not provide support for adaptive UIs. It restricts the mapping to a limited set of components, and it is not possible to switch between various presentations, validations at runtime, or to integrate other concerns. On the other hand, MetaWidget integrates transformations to multiple platforms. However, neither AspectFaces nor MetaWidget provides global optimization to derive optimal UI component selection and layout.

The first direction to consider for automated derivation of AUI is model-driven development, but it suffers from various disadvantages. First, the design of the model brings a development and maintenance overhead. Most likely, there is an application backend, and this model must be kept consistent manually. Second, models that need to capture different concerns might become complex. Multi-model integration can become an issue from the integration and performance perspective [16] when model-to-code transformation is at runtime, which is suggested by [9]. Third, the MDD does not solve problems with crosscutting concerns that can arise at model level.

## 2.3 UI description languages and UI delivery

This section focuses on selected examples of UIDL. The focus will be on those UIDLs that incorporate some level of abstraction from the target UI platform. Automatic CUI generation as destined in this thesis require platform-independent input instead of description tailored for a particular platform, e.g. Windows desktop. The input UI description language must effectively describe UIs that can be later automatically transformed for for currently requested target UI platform and current contextual conditions.

A User Interface Description Language abstraction can be seen on the *language level* or on the *UI content level*. On the *language level*, a single language can be used to describe UIs delivered to various platforms, but a particular UI described in such a language corresponds to a concrete contextual situation – typically the target platform. In the vocabulary of this thesis this level of abstraction corresponds to CUI. In the following text, UI description languages with this level of abstraction will be designed as CUI level abstraction UIDLs.

On the contrary, the abstraction on the *content level* allows single UI description to be used in many contextual situations. This abstraction level corresponds to AUI as defined above. Abstraction on the *content level* UI level requires, apart from trivial cases, context-sensitive automatic UI generation to derive CUIs that can be rendered on a particular target UI device.

*User Interface Markup Language* (UIML) [1] is an early language for platform-independent UI description. It is an XML-based language, that enables abstraction on the description level (AUI). UIML does not support the UI description on other levels. We can currently state that this language has been deprecated.

XML User Interface Language (XUL) [35] is a UI markup language developed by the Mozilla Foundation. It is based on XML. Using *XUL*, graphical UIs can be implemented similarly to *Web pages*. It also uses multiple existing Web technologies such as Cascading Style Sheets (CSS) [96], Java Script or Document Object Model (DOM) [73]. The abstraction level of XUL is rather on the CUI level, however, some level of context-independence of a single UI description can be reached using the CSS and run-time manipulation with the DOM.

MARIA [80] is a universal declarative UI description language that supports multiple abstraction levels. It was designed to support applications running in Service-oriented Ubiquitous environments. Abstraction levels considered by MARIA are Task level, Abstract level, Concrete level and Implementation level. The later three

levels corresponds to AUI, CUI, and FUI levels in terms of this thesis. MARIA extends the concept of Web services defined by WSDL language with annotations that enable the generation of UIs. Accordingly [81], the current trend is that concept of complex Web-services is being replaced by stateless RESTfull applications. This fact and description-complexity of Web-services (WSDL) prevents MARIA from broader practical use.

USIXML [58] is an XML-based UI description language. Similarly to MARIA, it supports the definition of UIs on multiple levels – Task and Concepts level, AUI level, CUI level. USIXML architecture considers multiple models that are used for UI transformation. There are Domain Model, Task Model, AUI Model, CUI Model, Mapping Model Context Model and Transformation Model. The authors suggest integration of these models into one formal and uniform format. On the contrary, this induces relatively high language complexity and therefore complicated adaption by new developers.

Extensible Application Markup Language (XAML) [25] is an XML-based declarative language for Microsoft .NET applications. XAML is widely used for Windows Presentation Foundation, Silverlight, and Windows Workflow Foundation applications. When used for UI specification, XAML defines structure of UI elements, data binding, and eventing. However, XAML language can be used for definition of UIs desired for different platforms, each definition corresponds to a concrete platform. In the terms of this thesis, UIs described by XAML correspond to CUI level.

Multimodal Interaction Markup Language (MIML) [3] focuses on a three-layered description of agent-based interactive systems. It defines dialogue patterns between humans and various types of interactive agents, including voice-dialogue systems or event human-resembling robots. MIML defines multimodal UIs on three different levels – task level markup language, interaction level markup language and platform level description. The *Task Markup Language* is an XML-Compliant language, each its document consists of two parts – head and body. This resembles the structure of an *HTML* document, but the semantic is substantially different. The *head* part basically contains a Context Model with two sub-models: UM and Device Model (DM). Each *body* part consists the information that should be presented to the user, followed by the information that should be retrieved from the user. There is also concept of information filtering according user's knowledge level. The interaction level markup language is modality-independent extension of VoiceXML language. The platform-independence of the MIML was archived using advanced binding on the lowest level. Using this concept task and interaction level are platform-independent.

The interaction level markup language corresponds to AUI in the terms of this thesis.

*Synchronized Multimodal User Interaction Modeling Language (SMUIML)* [31] is a UIDL for multimodal interaction. The authors presented nine guidelines for UIDLs for purposes of the multimodal interaction. In the scope of this thesis, we should notice requirements about the abstraction, the control over the fusion mechanism, the event management and adaptivity regarding to the CM and the UM in particular.

In this section we summarized selected UIDLs related to our approach. Table 2.3 shows the analyzed UIDLs, their abstraction levels, pros and cons. Although many of them bring interesting concepts, most languages that are practically used like *XAML* or *XUL* do not offer a satisfactory level of abstraction. In contrary, some languages, like *UIML*, enable UI definition only on the abstract level. This can cause issues if there is a need of manual modification of UIs resulting from context-sensitive transformation of this input model. The manual modification would require knowledge of any output language, which increases development efforts and amplifies probability of introducing an error.

Table 2.3: Comparison of UI description languages

UI markup language	Abstraction	Pros	Cons
UIML	AUI	AUI concept	Obsolete
XUL	CUI	Real-world use	Low abstraction, strong relationship to web
MARIA [80]	Task, AUI, CUI	Multiple abstention levels	Its connection to web-services
USIXML [58]	Task, AUI, CUI, Context, Domain, Mapping	Multiple abstention levels, system of transformations	Language complexity
XAML [25]	CUI	Widely used	Mostly for Windows platforms, only CUI level
MIML [3]	task, interaction, platform	Advanced binding	Possible information replication
SMUIML [31]	Dialog, AUI, CUI	follows guidelines for multimodal-interaction UI description languages	Lack of context-adaptivity

## 2.4 Context model

Context Model (CM) is an important attribute for the CUI generation when we want to adapt to current conditions of use. Common adaptations are regarding to attributes of the interaction device used, the user or the environment the interaction is carried out. In this section we analyze suitable context modeling methods.

Regarding user capabilities and preferences, most CMs currently strive to fit people with disabilities to standard systems, using various assistive technologies. The ability-based design [113] has been proposed as a response to this problem. It uses context-awareness to provide adaptations to user-specific abilities, instead of forcing users to use a specific assistive technology. In order to provide context-aware adaptations, there must be a context model. CM typically consist of models of user, device, and environment. According to [113], the problem with currently used context models is that they leverage the user disabilities rather than the abilities.

In the scope of this thesis, the UM is the most important component of the context model. However, other components of the CM exist. Most context modeling approaches define DM and EM. Another sub-models can be also defined, but often they are specific for a certain application domain.

In the following text we summarize approaches for user modeling. The original approaches emerged from rehabilitation engineering, where the human performance was measured and quantified in order to provide better adaptations of patients. The focus of rehabilitation engineering is much wider than just information and communication technologies (ICT). In this field the user models emerge typically from the medicine. The most common models are defined by the World Health Organization – The International Classification of Diseases [69] and International classification of functioning, disability and health [114]. From the point of view of UI design is interpretation of such models rather complicated. Influence of values in these user models (more or less medical data) to user interface is typically unclear.

The user context (UM) can be modeled using formats desired for information exchange. An example of widely used one is the *vCard* [22] file format – a standard for electronic business cards. Some basic information about user can be described by standard *vCard* properties – e.g. gender, spoken language or timezone. Despite this, for purposes of our work we need a more complex UM that can also describe properties important for the automatic adaption of UIs.

Another example of a context-modeling approach that focuses primarily on the user modeling is described in [103] as ETSI ES 202 746 draft standard. A number

of user characteristics and preferences that apply independently of any particular application can be described using this approach. Users should be able to specify their context-dependent needs in ways that require the minimum need to understand the individual applications. Properties relevant to adaptations based on the UiGE Pipeline are for example the preferred input (and output) modality, simple text (whether text-simplification should be used), symbols (whether symbols should be used to represent information), visual preferences like brightness, content-contrast or font-size. This approach represents a promising context-modeling method that could be adapted for purposes of the context-sensitive CUI generation. On the other hand, clearer separation of UM and DM properties would be necessary to simply the development and maintenance of CM instances.

Knappmeyer et al. [50, 51] presented a lightweight XML-based context representation schema called *ContextML*. Context information is categorized into scopes related to different entity types. *ContextML* uses REST-based [89] communication between framework components. The presented context representation method provides generic context representation and context exchange approach. The context is represented by entities. Each context state is called scope and provides a consistent context-instance at a specific moment. The proposed approach has been tested on two mobile platforms (iOS and Android) with following context providers: Location Provider, Civil Address Provider, User Profile Provider, Environment Provider, Time Context Provider, and Activity Provider. This context model can be organized in various abstraction layers from primitive scopes to high-level scopes. It is extensible – this method allows adding new scopes in runtime and in plug and play fashion. *ContextML* lacks the schematic strength of the Web Ontology Language (OWL) [64] or the Resource Description Framework (RDF) [49] based ontologies. Instead, it aims for light-weight context representation that can be used on mobile devices with constrained resources. The main advantage of this approach is the fact it supports a combination of context modeling, maintaining and exchange. However, an explicit relationship between context attributes represented by *ContextML* and UI attributes must be specified to enable the use of a similar approach in the framework of our approach.

Sheng and Benatallah in [97] propose *ContextUML* – a Unified Modeling Language (UML) based context modeling language for MBUID. UML is industry de-facto standard for design and development of software systems. The proposed context model is desired for development of context-aware Web services. The authors distinguish two types of context – atomic context and composite context. Atomic



contexts are low-level context instances that do not rely on other contexts and can be directly provided by context sources (e.g. sensors). On the contrary, composite contexts aggregate multiple contexts on the high-level to provide more abstract information. Authors further focus on context awareness modeling by introducing two mechanisms – context binding and context triggering. The current implementation of *ContextUML* is tailored for Web services.

Peißner et al. in [82] focus on individualization patterns for accessible and adaptive UIs. *MyUI* User profile variables include: visual acuity, field of vision, ambient light, ambient noise, language reception, language production, attention, processing speed, working memory, long term memory, Information and Communication Technologies (ICT) literacy, hand-eye coordination, speech articulation, hand precision, contact grip, last name, email address, preferred language, etc. From this list we can see that the user profile mixes different variables with low consistency. Some variables like ambient light or noise are not usually not included in the category of User Model (profile), instead they technically fit into the category environment model (alternatively into the device model). The influence of the proposed variables to the composed UIs is also not clear. On the contrary, *MyUI* supports runtime tracking of context changes and their reflection in the corresponding UI.

Razmerita et al. [88] focused on ontology-based user modeling for knowledge management systems. Their architecture used three different ontologies: user ontology, domain ontology and log ontology. The authors mentioned user properties and characteristics like identity, email, address, competencies, cognitive style, and preferences. In general the user model is structured according Leaner Information Package specifications [43], user model contains eleven groupings: Identification, Qualification, Certification, Licenses, Accessibility, Activity, Competence, Interest, Affiliation, Security Key and Relationship. The presented approach incorporates explicit part of user model that is maintained by the users themselves. An implicit part of user model uses various techniques to encourage users to codify their experience. The system uses user behavior analysis and related heuristics to codify the level of user activity, level of his/her knowledge sharing etc. This approach also incorporates social and gamification [28] aspects to encourage users to be more active. Technical implementation of this approach uses a *OntoUM* server that stores data in the RDF [49] format.

Kaklanis et al. in [47] focus towards the standardization of UMs for simulation and adaption purposes. The authors propose Virtual User Modeling and Simulation Standardization (*VUMS'*) project cluster aiming to develop an interoperable UM.

Table 2.4: Comparison approaches for context modeling

Context model	Sub-models	Pros	Cons
<i>vCard</i> [22]	UM	Established standard	Limited information use-able for UI adaption
ETSI ES 202 746 [103]	Primarily UM, partially DM	Relevant properties in UM	Interconnection of CM properties
ContextML [50, 51]	DM, UM, EM	Both context representation and context exchange, lightweight context representation	No direct relationship to UI attributes
ContextUML [97]	Not specified by design (UM, EM in example)	Atomic and composite contexts	Tailored to web, does not specify context structure
Peißner et al. in [82]	UM	Clear relationship to UI attributes	Model complexity, unclear relationship to UI
Razmerita et al. [88]	UM	Complex multi-factor model	complicated
VUMS project [47]	UM	Focus of people with disabilities	Leverage users' limitations
Dynamix [12]	DM, EM	Context sensing, device integration, Application Programming Interface (API)	To-date restricted to Android

This UM is able to describe both able-bodied people as well as people with various kinds of disabilities. The VUMS project cluster consists of four Framework Program (FP) 7 EU projects – VERITAS, MyUI, GUIDE, and VICON.

Carlson and Schrader in [12] suggest a new community-based approach for context-aware computing. Dynamix is a lightweight background service running on a user's mobile device. It models context information from the environment using user's mobile device itself as a sensing, processing and communication platform. A simple to use API is exposed to applications that request context support. Dynamix supports automatic discovery, downloading and installation of plug-ins required for a given context-sensing task.

In this section, we presented several approaches for context modeling. Those approaches are listed in Table 2.4. The original approaches like [114] emerged from rehabilitation engineering. According to [113], the main issue is that most current context modeling approaches leverage user's disabilities rather than their abilities. There are also context modeling approaches that are based on complex ontological models. However, these approaches can model complex contextual instances in a particular domain, they induce substantial issues from the perspective of develop-

ment and maintenance efforts due to their complexity. For purposes of this thesis, we need a context model that is easy to maintain on one hand and can model level of user specific abilities in the same way as their limitations on the other hand. A promising approach that inspired our solution is draft standard ETSI ES 202 746 described in [103]. However, it provides an inspiration for relevant CM properties, the corresponding CM structure contains unnecessary interconnection between UM and DM properties that could complicate the development and maintenance of CM instances.

## 2.5 Automatic UI evaluation

An automatic UI generation method approach requires a method how to automatically assess the quality of the generated UIs in order to provide optimal results. This section focuses on the promising state of the art methods available that could be potentially used.

In accordance with the aims of this thesis, an important objective is to generate UIs that will be maximally useable. According to ANSI 2001 [79] and ISO 9241 part 11 [44], the dimensions of usability are effectiveness, efficiency and satisfaction. For the process of automatic UI generation, we need a method that will provide automatic UI evaluation from the usability perspective. This section focuses on existing methods that can meet such a requirement.

There are several laws and rules of thumb that are used for UI development and evaluation for decades. For example Hick's Law [90] puts into the relationship the number of available choices and decision time, whilst Fitts' Law [62] model the duration of the act of pointing. There are also rules emerging from psychology that suggest how to compose the UI structure (layout), most notably the laws of organization in perceptual forms, known as Gestalt rules [111].

The Nielsen heuristics [75] consist of ten general principles for interaction design. The heuristics were derived by factor analysis of 249 usability problems to provide maximum explanatory power. Due to their general manner, in the original form, those heuristics require a human expert to be present for both the design as well the evaluation of an interactive system.

Okada et al. [76] describe two methods for automated evaluation of usability and accessibility of web pages. Usability is evaluated using analysis of logs resulting from user interaction. The author focus mainly on the effectiveness by comparing real logs with logs resulting from the ideal (desired) interaction. Accessibility was

evaluated using machine learning methods.

Gimblett and Thimbleby [38] describe an evaluation method that uses a theorem discovery method to automatically find and check usability heuristics. They automatically and systematically look for sequences of user input that are equivalent (or almost equivalent) in their effect on a system. Authors state that such almost equivalent actions that result in different outcomes are source of potential errors and user confusion. This method requires a complete UI to be used in the full extend, however, it can be also used for the iterative development. AUI as defined for purposes of this thesis already contain available actions, hence, this method is not useable for CUI derivation. From our perspective, it can be used for optimization of AUI derivation from other input models.

Fainer and Andrews [34] propose the usability markup language – *UsabML*, which defines a structured reporting format of usability evaluations. A web-based system called Usability Reporting Manager (URM) can handle formative usability reports described in *UsabML*. Discovered issues can be directly imported into standard issue tracking systems. However, *UsabML* can be useful for structure formative usability reports, the proposed tool does not offer real automated usability evaluation.

Chattratchart and Lindgaard [19] describe a comparative evaluation of heuristic-based usability inspection methods. They compare classical heuristic evaluation (HE) based on the Nielsen heuristics [75] with HE-Plus and HE++ heuristics. They conclude that both HE-Plus and HE++ outperformed HE in terms of effectiveness and reliability. The main reason is that HE-Plus and HE++ support focusing on problem areas.

Sauro and Kindlund [92] describe a method to standardize usability metrics into a single score – the summated usability metric (SUM). The SUM provides one continuous variable for summative usability evaluations that can be used in hypothesis testing and usability reporting. The user satisfaction was measured using questionnaires including questions on task experience, ease of task, time on task and overall task satisfaction.

Cassino and Tucci [13] describe an approach to evaluate interactive visual environment based on SR-Action Grammars formalism. Using their approach it is possible to aid the developer to create applications that automatically respect a significant number of usability rules. They propose *VALUTA* - Automatic tool for usability verification at the abstract level. The system takes as input an interactive visual application, generates related formal specification and automatically performs the implemented usability checks. The authors implemented a verification of a set of

Nielsen heuristics [75]: completeness, correctness, aesthetic and minimalist design, user control and consistency. The current version of the *VALUTA* tool is tailored for Web-pages.

Kurniawan and Zaphiris in [54] present a set of UI development guidelines focuses on elderly people. The proposed guidelines are divided into eleven categories focusing on: target design, use of graphics, navigation, browser window features, content layout design, links, user cognitive design, use of color and background, text design, search engine, and user feedback & support.

Seffah et al. [95] provide an evaluation of currently used usability measurement methods. Upon their analysis, they provide an consolidated hierarchical model of usability measurement. The authors introduce a Quality in Use Integrated Mea-

Table 2.5: Comparison of UI evaluation methods

Method	Focuses-on	Pros	Cons
Nielsen heuristics [75]	Usability of interactive systems in general	General, famous, used for heuristic evaluation	Too general for automatic UI evaluation
Okada et al. [76]	Web usability and accessibility	Automatic evaluation of effectiveness	Requires user interaction
Gimblett and Thimbleby [38]	Usability, error prevention	Can estimate errors caused by ambiguous input	Require complete UI, above AUI level
UsabML [34]	Usability reporting language - <i>UsabML</i>	Formalized description of usability reports	Not an automatic tool
HE-Plus and HE++ heuristics [19]	Evaluation of Heuristic Usability evaluation methods	Focusing on problem areas improves evaluation	Not an automatic evaluation tool
Sauro and Kindlund [92]	Usability metrics	Introduction of summated usability metric (SUM)	Uses questionnaires (requires user)
VALUTA tool [13]	Automatic usability evaluation on abstract level	Automatic solution	Tailored for web pages
Kurniawan and Zaphiris [54]	Usability guidelines for UI design for elderly	Example of user-specific guidelines	Only one user group
Seffah et al. [95]	Quality in Use Integrated Measurement (QUIM) model	Integration of usability evaluation methods	Very complex, require human evaluation

surement (*QUIM*) model that brings together usability factors, criteria, metrics and data described in various standards in a consistent way. *QUIM* bring improvements in usability measurement planing as well as calculating metrics of overall usability goals. However, *QUIM* archives integration of multiple usability metrics, most methods require being carried out by human experts.

In this section, we summarized various approaches related to automatic usability evaluation, see Table 2.5. Some approaches bring only minor level of automation, e.g. they help with construction of consistent usability reports – [34, 92, 95]. Many approaches require the presence of human experts – [95, 19] or are based on measured real user activity [76, 92]). Few approaches that offer true automation of usability estimation exist – [13, 38]. However, even those approaches are not suitable for CUI usability estimation, at least not in their current form. VALUTA tool [13] focuses on discovery of potential usability problems of web applications. Method by Gimblett and Thimbleby [38] can discover potential issues caused by ambiguous input, however, it needs complete UI and functional description on a high level of abstraction.

The evaluation is an integral part of the UI creation process. A best practice for effective development of usable UIs that respect needs and preferences of their target users is the *User Centered Design (UCD)* [2] process. Generally, the UCD is a design process in which end-users influence how a design takes shape. Various observation, design and evaluation methods can be used for the UCD, including user modeling using a *Personas* technique [66]. UCD process has been standardized by the International Standardization Organization (ISO) [30]. UCD typically consists of four stages that are iteratively repeated several times to evolve product that fits the needs and preferences of its users. UCD should be considered during development of related development support tools, see appendix A. Also, use of the UiGE Pipeline could make the UCD UI development process more effective by simplifying UI prototyping and evaluation.

## 2.6 Conclusion

This chapter summarized approaches and methods related to the objectives of this thesis stated in section 1.2. Firstly, in section 2.1 we focused on related approaches for automatic UI generation in general. Many approaches bring interesting methods suitable for automatic UI generation, like combinatoric optimization brought by Supple [37], UI consistency preservation (Uniform [72]) or aspect-based transforma-

tion (AspectFaces [16]). Supple [37] brought an inspiration for the CUI optimization method described in this thesis, that is also based on the combinatoric optimization. However, most related approaches have limitations emerging from limited UI adaptation they offer on one hand and complicated development and maintenance caused by complex coupled input models on the other hand.

Section 2.2 focused on related methods to effectively derive AUIs from other input models. Analysis of several MDD based approaches resulted into concluding that they induce problems during adaptation and evolution management. This statement is in accordance with [67]. The code-inspection and aspect-oriented transformation turned out to be a promising method for derivation of AUIs in the case of data-oriented applications. Section 4.3 describes the theoretical background of the use of code-inspection and aspect-oriented transformation for purposes of AUI derivation. A practical application of this approach is described in chapter 7.

Our survey of UI markup languages is described in section 2.3. It has shown that several universal UIDLs does not offer a satisfactory level of abstraction. On contrary, some languages allow UI definition only on an abstract level. A UIDL that uses a similar structure and is based on similar concepts on both abstract and concrete level might simplify our approach from the developer's perspective.

The Context Model is an important attribute for the CUI generation, section 2.4 focuses on survey of context modeling methods related to our approach. The outcome is that various context-modeling approaches are suited to a particular domain (e.g. [22]), many context modeling approaches are also based on complex ontologies. This complexity can cause development and maintenance difficulties in case of complex real-world system. For purposes of this thesis, we need a CM that is easy to maintain on one hand and can model users' specific abilities in the same way as their limitations on the other hand.

Last but not least, section 2.5 focuses on methods for automatic evaluation of UIs. Our CUI generation approach needs a method to assess the quality of the generated UIs in order to provide optimal results. The outcome of our survey of is that most methods provide some level of automation, but still require presence of human experts or need to track the user activity while using a real application. In case of methods based on heuristic evaluation only those that focus on a specific application domain (e.g. web) provide a satisfactory level of automation. Our approach should allow general optimization based on simple metric (e.g. the number of steps to carry out an action). For specific purposes, specific heuristic rules can be used to provide better results from the UI usability perspective.





# Chapter 3

## UI Description and Delivery

This chapter focuses on the realization of the UI delivery method. More precisely, we focus on the situation when UI described in common CUI language are delivered to various client platforms. In our case, these client platforms can significantly differ in operating systems, screen sizes, supported interaction modalities, etc. For an effective UI generation and delivery it is convenient to have a single internal UI description language that can be interpreted on different platforms.

The approach described in this thesis in the form of Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline), including the automatic CUI generation method, Context Model, and AUI derivation from other input models is implemented as the *UIP Platform*. A necessary foundation for these more advanced methods is the CUI delivery method described in this chapter. Figure 3.1 shows the scope of the UI delivery method as defined for purposes of this thesis. The steps of the UI generation pipeline realized by the method described in this chapter are marked by the dashed rectangle in Figure 3.1. The primary focus is on delivery of CUIs to various client platforms and their final rendering (transformation into the FUI).

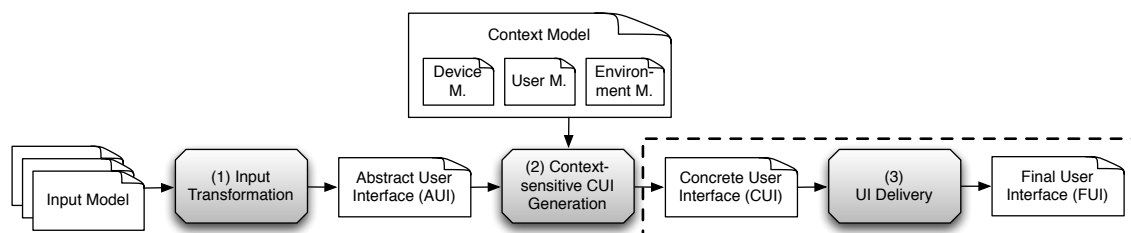


Figure 3.1: UI delivery method in relation to Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

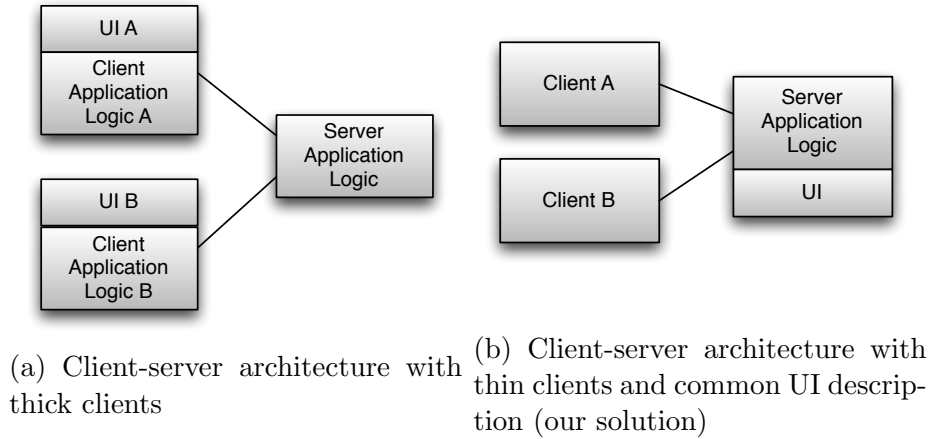


Figure 3.2: Client-server architecture with thick and thin clients

### 3.1 UIP Platform

The *UIP Platform* has been designed to provide an effective foundation for client-server applications including heterogeneous client platforms and to support ubiquitous computing [4]. It is based on client-server architecture (see Figure 3.2). Whilst architecture of multi-platform client-server application with thick clients (see Figure 3.2a) introduces splitting of the application logic between various client platforms and a common server part. Client platforms also often use different UI description languages. This requires additional effort from developers to deal with multiple heterogeneous platforms.

The *UIP Platform* incorporates platform-independent UI description by means of *UIP Protocol*. In the case of *UIP Platform* (see Figure 3.2b), there is common application logic on the server. Concrete UI definition (CUI) can be delivered and rendered on any supported client platform. A clear separation of presentation, model and application logic (which resides on the server by design) is ensured by the platform design.

*UIP Platform* also defines communication between client(s) and the server. As shown in Figure 3.3, only events that are later handled by the server side application logic are propagated in the direction from a client to the server. In the opposite direction the server can push interfaces (description of UI structure) and models (data that are represented to the user through the UI) to a client. As soon as a model is updated on the server side, this update is propagated to all relevant clients. Such update is then instantly reflected in the UI using standard data binding mechanism, see [65].

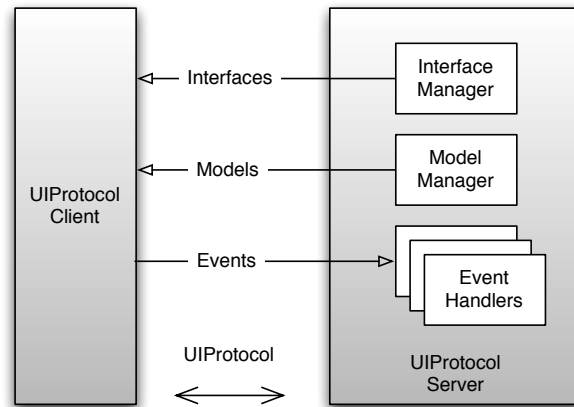


Figure 3.3: UIP Protocol communication

*UIP Protocol* is part a of the *UIP Platform*. As depicted in Figure 3.4, *UIP Protocol* consists of *Event Protocol* and *UI Protocol*. The *Event protocol* describes the communication using *Event Documents*, which is typically used for communication in the direction from *UIP client* to *UIP server*. Additionally, *Event Protocol* is used for peer-to-peer communication between other distributed *UIP Platform* components. One example is the communication between the *UiGE* and a *UIP client* to determine platform-dependent UI element dimensions and appearance at the client side. Another example is the communication between the *UIP Visual Editor* (see section 4.2) and *UIP clients* to determine platform-dependent visualization of UI elements.

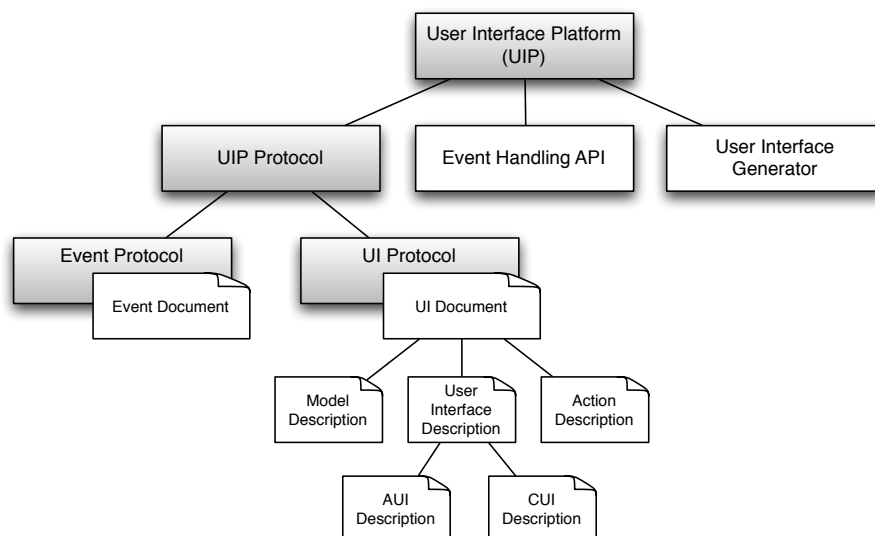


Figure 3.4: UIP Platform conceptual components and protocols

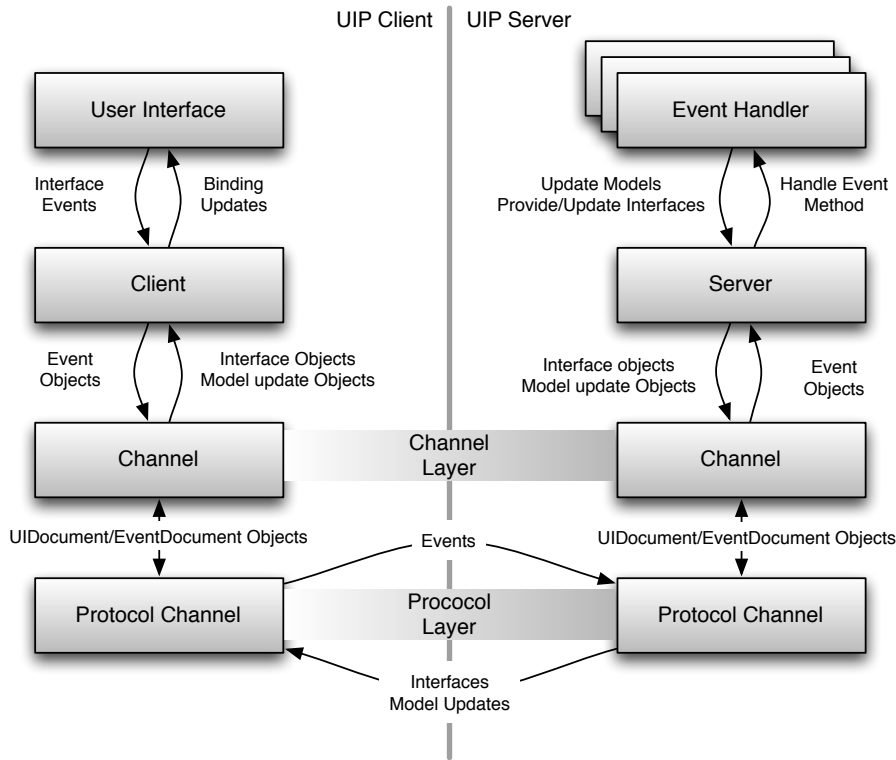


Figure 3.5: UIP Protocol reference architecture

*UI Protocol* describes communication using *UI Documents*. A *UI Document* (see Figure 3.4) describes data *Models*, and *User Interface* descriptions. *User Interfaces* descriptions sent to a *UIP Client* are always CUIs. However, *User Interface* Description can also have the form of an AUI that can be transformed into a CUI using the *UiGE* – see chapter 6.

Figure 3.5 depicts the *UIP Protocol* reference architecture. The *Protocol layer* is abstracted from the *Channel layer*. An object representation only is exchanged inside a *UIP Client* and a *UIP Server*. The *UIDocument/EventDocument* objects can be represented differently on the *Protocol layer*. There is an XML-variant, a json-variant [23], and a binary variant of the protocol.

Communication in the direction from client to server is carried out as follows: Firstly, user’s interaction with UI rendered on a UIP client results into the generation of *Interface Events*. These events are encapsulated in *Event Objects*. In the next step, communication channel transforms them into the *EventDocument Objects* that can be transported on the protocol layer. The protocol variant (XML, json, binary) is already negotiated with the server during the connection handshake. On the server side, *Event Document* objects are transformed into an internal object representation

– *Event Objects*. *Event Objects* are handed over to corresponding *Event Handlers* that implement application logic in the UIP Platform.

In the opposite direction from the server to the client is the communication triggered by *EventHandlers*. An *EventHandler* can update data *Models*, and provide (update) user *Interfaces*. Similarly to UIP client, *Models and Interfaces* have internal object representation. For purposes of the communication channel are object transformed into a corresponding *UI Document* form.

## 3.2 UIP clients

Various UIP clients have been implemented as part of the *UIP Platform*. They differ in their software platforms (Java, .NET, PHP, ASP, Adobe Flash, iOS or Silverlight) as well as in the type of supported device (PC, generic web, Tablet PC, Smartphone, or even multitouch tabletop). The *UIP Platform* then supports a wide range of clients with different capabilities and software foundation. Using the capabilities of *UIP Protocol* a single UI can be rendered on any supported *UIP Client*.

Figure 3.6 depicts reference architecture of *UIP client*. It consists of the *Common UIP client core* and *UIP Client Platform extensions*. The *Common UIP Client Core* is consistent for all supported platforms. It is designed to be maximally reusable among all supported platforms. Whereas *UIP Client Platform extensions* represent part of client implementation that corresponds to a particular client platform.

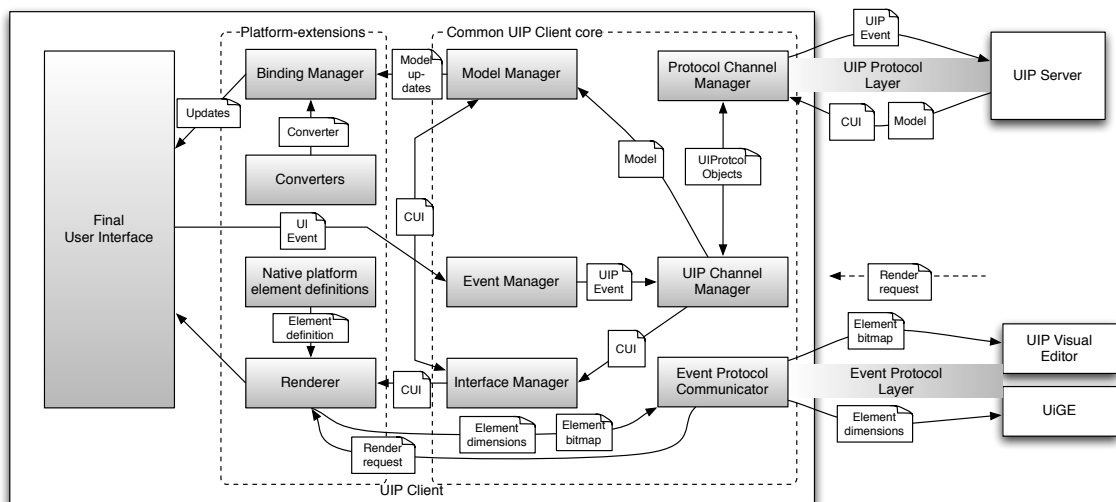


Figure 3.6: Reference architecture of *UIP Client*

### 3.2.1 Common UIP Client Core

The *Common UIP Client Core* communicates with the *UIP Server* on *UIP Protocol Layer* using *Protocol Channel Manager*. The representation of *UI Protocol Documents* on this layer has multiple variants, namely: *XML*, *binary* and *json*. The standard variant is *XML* that must be supported by any *UIP Client*. In the direction from *UIP Client* to *UIP Server*, the *UIP Protocol Layer* transfers only *UIP Events* that are handled by application logic persisting on the *Server* or forwarded to other integrated platforms like smart-home hub by means of *UIP Server* application logic. In the opposite direction, the *UIP Server* transmits *User Interface* definitions, and *Data Models*. *Protocol Channel Manager* transforms *UI Protocol Documents* transferred on the *UIP Protocol Layer* to internal object representation and vice versa.

*UIP Channel Manager* encapsulates most of the *UIP Client* internal logic. It distributes *UIProtocol Objects* provided by *Protocol Channel Manager* to corresponding *UIP Client* components. Further, it is responsible for initialization of these components and for connection handshake.

*Model Manager* is responsible for maintaining data *Models* sent from the *UIP Server*. It exposes data contained in the *Models* to the *Binding Manager* that updates UI element properties in the platform specific *FUI*. Model updates are not necessarily instant, they can be delayed and interpolated. The *Model Manager* implementation integrates interpolators that implements interpolated model updates. Among others, interpolated updates can be used for UI animations.

*Event Manager* is responsible for handling *Events*. *Events* typically originate in user actions while interacting with the *FUI*. Also, events can be triggered by some components of a *UIP Client* (e.g. connection event, interpolation finished event, model request event, or interface request event). Generated events are propagated to the *UIP Server* by *UIP Channel Manager*.

*Interface Manager* handles *CUIs* provided by the *UIP Server*. The *CUIs* are maintained in cooperation with *Model Manager* – a specific *Model* is designed for storing *CUIs* on a *UIP Client*. *Interface Manager* provides the *CUI Renderer* with *CUI* descriptions.

#### 3.2.1.1 UIP Client Event Protocol Communicator

The *Event Protocol Communicator* is an additional component of *UIP Client* that is not required for its basic function. Its purpose is to provide client-specific in-

formation about FUI elements, even in a case that a *UIP Client* is not connected to any *UIP Server*. In the current implementation of *UIP Platforms* there are two purposes for *Event Protocol Communicator* on *UIP Client*:

- Providing information about platform specific UI element dimensions for UiGE.
- Providing geometrical dimensions and raster image representation of platform specific UI elements to UIP visual editor.

### 3.2.2 UIP Client Platform Extensions

*Platform Extensions* contain platform-specific *UIP Client* components that are typically re-implemented for each client platform to be supported. These extensions have usually the form of a module of more generic *UIP Client*, i.e. only one *UIP client* needs to be implemented for each development platform like *.NET* or *java* whilst each UI platform like *.NET Silverlight*, *.NET WPF*, or *Java Swing* requires its own implementation of *Platform Extensions*.

Basic component implemented for each UI platform is *Renderer*. In terms of this thesis, it is responsible for the transformation from *CUI* described in *UI Protocol* into a platform-specific *FUI*. As the FUI is rendered using platform-specific UI components, renderer must map *UI Protocol* properties to platform-specific properties (e.g. element-width, font-size, list of combo-box options etc.) of individual FUI components. Also platform specific event triggers (e.g. mouse click, finger tap, swipe left gesture etc.) must be mapped to *UI Protocol* event triggers.

In order to provide modular approach, *UIP Client Platform Extensions* contain *Native platform element definitions* that represent mapping between *UI Protocol* CUI elements and containers and their platform-specific FUI implementations. These extensions are typically loaded dynamically, *UIP Client* provides *UIP Server* with actually supported components during the connection handshake. Advanced components of *UIP Platform* like *UiGE* can adapt its functionality to set of UI elements supported by a current *UIP Client*.

The *Binding manager* is responsible for instant connection between the data *Model* properties and properties of native UI components of the *FUI*. Values of UIP properties represented as strings are type-weak, for most UI platforms it is necessary to convert these values to values of native UI element properties of various types. Set of platform-specific *Converters* needs to be implemented as part of a *UIP Client Platform Extension* to provide this conversion.

### 3.3 UIP Server

The *UIP Server* is the a central component of the *UIP Platform* responsible for providing *UIP Clients* with user *Interfaces*, data *Models* and handling received *Events*. It can serve multiple *UIP Clients* at once and provide them with independent content as well as perform complex *Model* and *Interface* updates for all connected clients. Architecture of *UIP Server* is depicted in Figure 3.7.

The main components of the *UIP Server* are the *UIP Server Core*, the *UIP Applications* and the *UIP Server Extensions*. The *UIP Server Core* contains main components that are responsible for the basic server runtime functions (configuration, logging etc.), communication with *UIP Clients*, management of content described by *UIP Applications* and management of *UIP Server Extensions*.

Similarly to the *UIP Client*, the communication channel abstracts from actual *UI Protocol* representation on the physical communication channel. Core component responsible for server runtime is *UIP Channel Manager* that also manages communication between *UIP Clients* via *Protocol Channel Manager* and the rest of *UIP Server* components.

The *Model Manager* is responsible for providing *UIP Clients* with data *Modes* and their updates. *Model* updates results from application logic or from external sources connected to the *UIP Server*. As *Modes* complexly affects *UIP Client* behavior, it is necessary to distinguish between *Model* versions for particular *UIP Clients*.

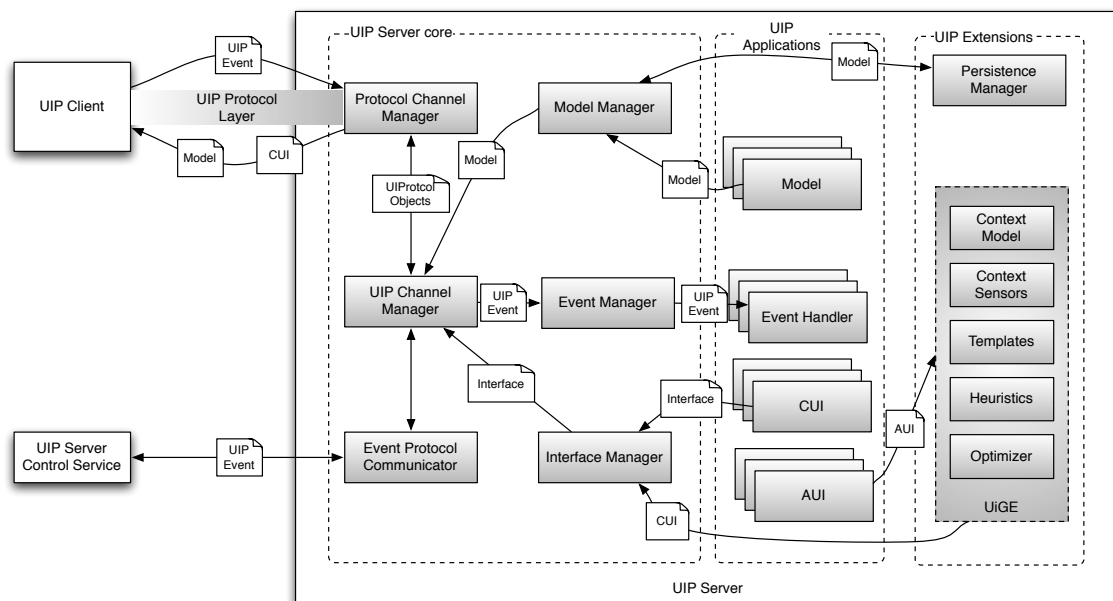


Figure 3.7: Reference architecture of UIP server



*UI Protocol* also defines so called *Model variants* that enables *UIP Server* to select *Model* sufficient for a particular situation (e.g. language mutation). *Model variants* are used in front of all for purposes of internationalization(i18n) when each model variant contains particular language constants. Initial data *Modes* are stored in *UIP Applications*. During the *UIP Server* runtime, the *Model Manager* contains actual data. Only specific (persistent) *Model* updates are saved as back to a *UIP Application*. *Models* can be updated from *Event Handlers*, in this way, the application logic on server affects behavior of particular *UIP Clients*.

*Event Manager* deals with *Events* received from *UIP Clients*. *Events* are typically propagated to *Event Handlers* defined in *UIP Applications* that represent the application logic. Specific events are handled by the internal logic of *UIP Server*, for example *connection event*, *model request event*, *interface request event* etc.. *UIP Server* exposes an *Event Handling API*, see Subsection 3.3.2. Using this API, *Event Handlers* in various programming (or scripting) languages can be defined. Currently there are *Event Handlers* in .NET, Java and ECMA Script.

*Interface Manager* is responsible for providing *CUIs* to *UIP Clients*. *CUI* definitions are either part of *UIP Applications* or generated by the *UiGE*. In the following text we describe a method how to program *UIP Server – Event Handling API*.

### 3.3.1 Event Handling API

The *UIP* architecture places the application logic completely on the server. This fact simplifies both application development and maintenance.

*UIP Platform* defines the *Event Handling API* as depicted in Figure 3.8. Initial version of this concept was presented in [other1]. Each *Event* dispatched to the *UIP Server* is propagated to the *Event Manager*. Besides special cases, each *Event* is propagated to an *Event Handler* – an instance of a class implementing *IEventHandler* Interface. A corresponding *Event Handler* is selected using the *class* attribute of an *Event*.

A method called *HandleEvent* is invoked for the selected *Event Handler*. This method is provided with three objects as parameters – *Client Object*, *Server Object* and *Event Object*. Using these objects, the program code implementing an *Event Handler* can manipulate *Data Models* as well as get necessary information. *Event Object* contains *UIP Properties* attached to a particular *Event*. These properties can be either defined in an *Interface* definition (e.g. additional parameters) or attached automatically by a *UIP Client* (e.g. class of element that invoked the *Event*).

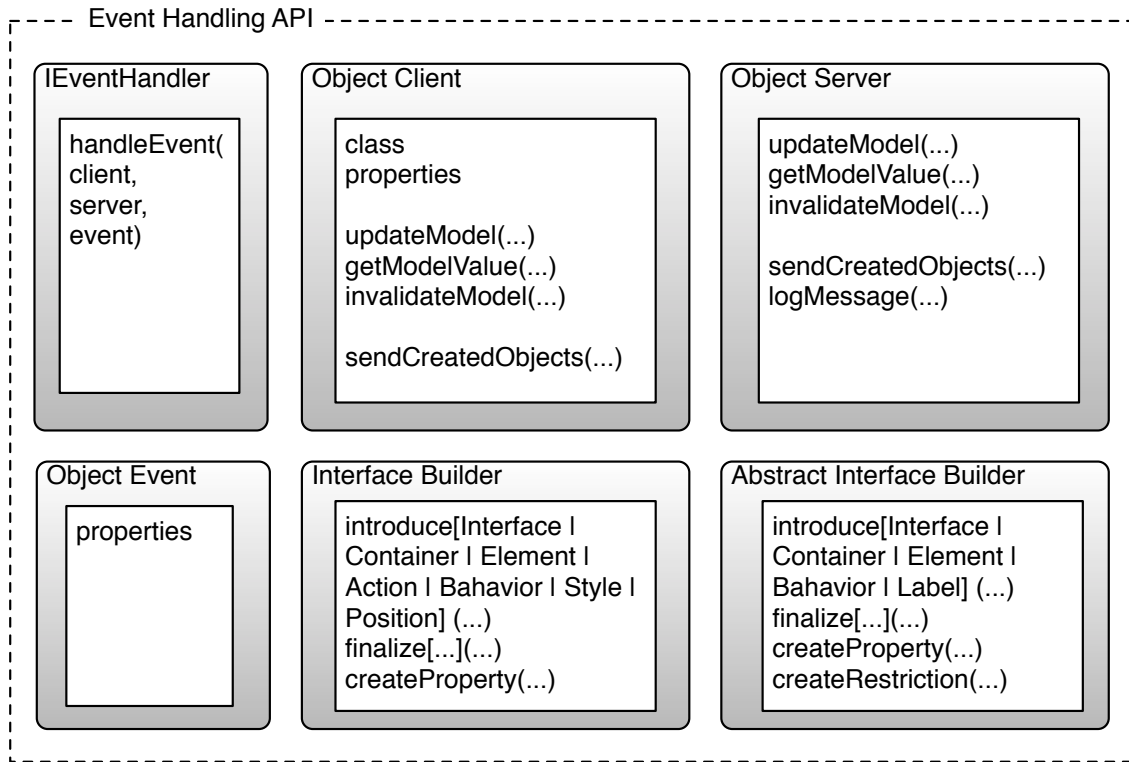


Figure 3.8: Simplified scheme of *Event Handling API*

Both *Client Object* and *Server Object* have methods to update *Model* and to get information stored in a *Model*. The main difference is that the *Client Object* manipulates with data related to a corresponding *UIP Client*, whilst the *Server Object* can manipulate with common data, or with data of other connected *UIP Clients* using an advanced client selection mechanism. There are three basic model update types – *Complete*, *Partial*, and *Persistent*. By setting a corresponding *Update Type*, it is possible to update only one property in a *Model*, completely replace a *Model* with updated properties or to perform an update that is permanently stored (into a corresponding *UIP Application* or into a database).

*Builders* are also part of the *Event Handling API*. The *Builders* are classes that can be used to create new *UIP Interfaces*. There are two types of *Builders* – *Interface Builder* to generate CUIs and *Abstract Interface Builder* to generate AUIs. These structures can be used to generate new AUI and CUI structures during the *UIP Server* runtime. *Client and Server Objects* have a method called *sentCreatedObjects* that can be used to propagate newly generated UI definition to particular *UIP Clients*.

### 3.3.2 UIP Application

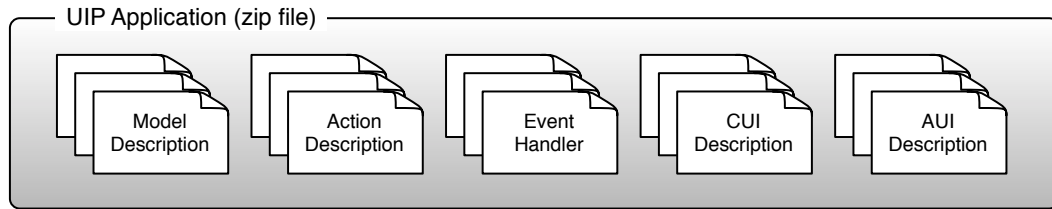


Figure 3.9: Components of a *UIP Application*

In the text above, we defined particular components of the *UIP Platform*. An application consists of multiple source-code documents that should be kept together in one structure. A *UIP Application* is a container for *Models*, *Interfaces (CUIs and AUIs)* and *Event Handlers*. Figure 3.9 illustrates individual components of a *UIP Application*. Typically, the *UIP Application* is implemented as a compressed folder (zip), with defined internal structure. *UIP Server* can be configured to load selected *UIP Applications*. A caching mechanism can be employed in order to decrease starting time of the *UIP Server*.

## 3.4 Conclusion and Contribution

This chapter described foundations of the *User Interface Platform (UIP)* that enables delivery of UIs to various types of client devices that use different UI platforms. We see the major contribution to the field by methods introduced in this chapter, namely by following:

- *Centralization of application logic on the server side.* Seen separately, this feature is not unique in comparison to the state of the art. However, as a strict requirement, it simplifies development for multiple client platforms with different programming languages.
- *Use of native UI elements.* A single UI description can be rendered on target platforms with various capabilities using native UI components (widgets).
- *Platform extensibility.* On the serve-side, the *UIP Platform* can be easily extended with additional components. E.g. *UiGE* CUI generator as the subject of this thesis is also an extension of the *UIP Server*.

- *Platform modularity.* As *UIP* uses layered model architecture as depicted in Figure 3.5, it is possible to replace individual layers by another implementation. E.g. apart from XML variant, other variants like binary (UIP-b) or JSON [23] (UIP-j) exist.

The *UIP Platform* as described in this chapter serves as a solid foundation for many applications, e.g. GraFooSha described in chapter 9 from the *UIP Platform* uses only methods described in this chapter. However, complex adaptations of CUIs, require automatic CUI generation (see chapter 6) that uses *Context Model* (see chapter 5).

# Chapter 4

## Input for User Interface generation

This chapter focuses on input for Context-sensitive automatic UI generation as defined above. Figure 4.1 shows components described in this chapter in the scope of general UI generation pipeline. Firstly, actual AUI structure and related *UI Protocol* version for AUI description are defined. Further, selected input methods that enable AUI derivation are described in greater detail.

Figure 4.2 depicts various input methods that can be used for AUI derivation. Although it is possible to specify input AUIs manually (Figure 4.2-a), in many cases it is useful to derive from another input model. One possibility is to use *UIP visual editor* (Figure 4.2-b) that enables interactive AUI specification in a graphical manner. Details about *UIP Visual Editor* are in section 4.2. For data-oriented applications, *Application audit (Code Inspection)* (Figure 4.2-c) can be used for very effective AUI derivation, see section 4.3. Finally, it is possible to transform another input models into a *UIP* AUI structure as depicted in Figure 4.2-d. Transformation of selected input models is described in section 4.4.

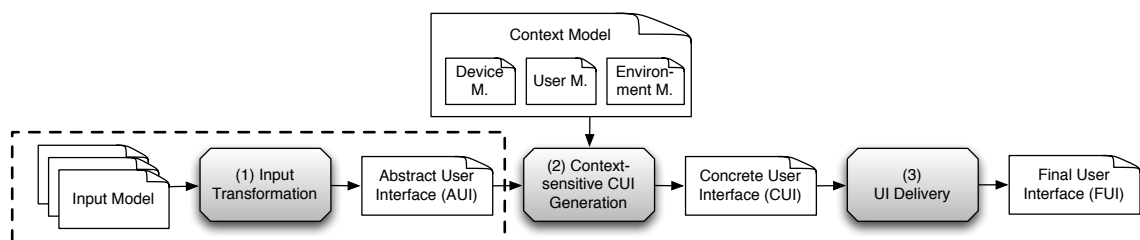


Figure 4.1: Input transformation scope in the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

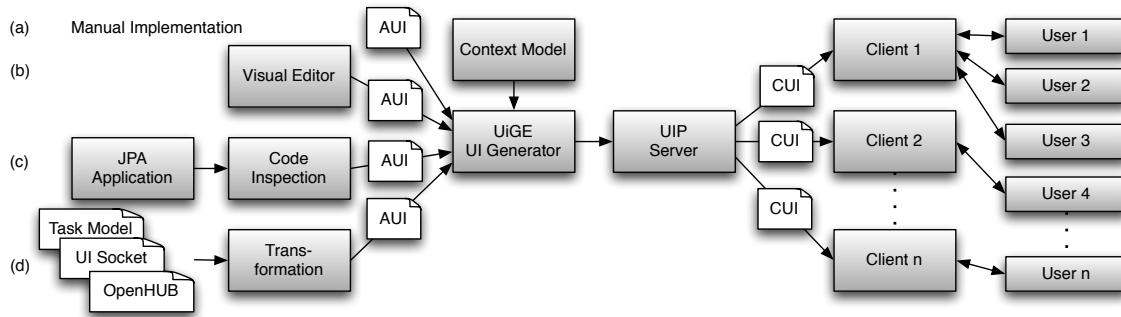


Figure 4.2: Input Methods

Figure 4.2 also shows how the UIP User Interface Generator (UiGE) provides various differentiating client UI platforms with context-adaptive UIs tailored to the needs and preferences of various users. This chapter concludes by enumeration of important contributions related to *UIP AUI* design as well as the statement of its limitations.

## 4.1 Abstract User Interface

General purpose of an AUI is to define UI structure in a context-independent manner. A single AUI can be then transformed into many different variants of CUI using context-sensitive CUI generation. Example of the hierarchical structure of an AUI is shown in Figure 4.3. The root node of an AUI tree is an *Interface*. Using the *class* attribute, it uniquely identifies an AUI in the scope of a *UIP Application*. A particular *AUI Interface* can be nested into another AUI interface or even into the CUI interface when necessary.

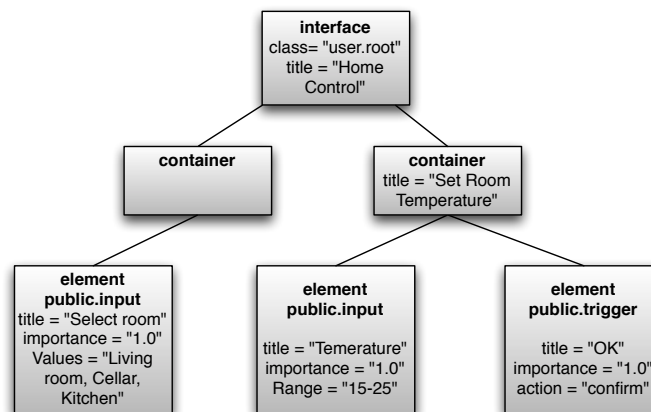


Figure 4.3: Example AUI structure

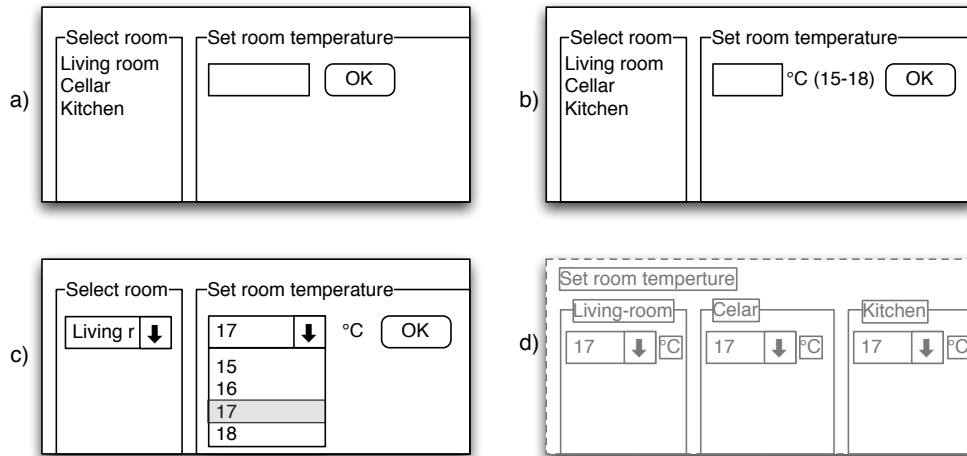


Figure 4.4: Functionally equivalent CUI's

The purpose of the *Container node* is to aggregate one or more child nodes into one structure. An *Element node* corresponds to a single interaction element in the resulting CUI structure. Figure 4.3 shows the example structure of an AUI that represents control of a home heating system. Examples a) to c) represent functionally equivalent CUIs that can result from the process of automatic CUI generation as described in this thesis. Example d) in Figure 4.3 represents a structure that corresponds to a different AUI structure, although even this CUI is still functionally equivalent to remaining three examples.

Similarly to the *UIP CUI*, an AUI is typically represented using the *XML*, although there are other more data-efficient variants. The AUI source XML code that corresponds to the example home heating system shown in this chapter is in the appendix B.1.

Any type of AUI node can be accompanied by *Label Node*. In the case of the AUI, the *Label Node* reference necessary information to provide an appropriate concrete label related to an element as a result of automatic CUI generation. Apart from the short text label, this element can contain also a longer description of element meaning or graphical mark of the element. *Label Node* also incorporates information that is provided to the user in for purposes of data-validation. Information for validation includes a label that describes extend of expected values in a human readable form as well as a label that should be shown in case that the validation criteria are not followed.

*Element nodes* can be additionally supplemented with *Restrictions Node* and *Behaviors Node*. Basically, *Restrictions Node* expresses the limits of element values,

in our case possible temperature range that can be set to the home heating system. *Behaviors Node* corresponds to *UIP Events* to be send as a result of interaction actions corresponding to a particular *Element*.

## 4.2 UIP visual editor

The *UIP Visual Editor* can be used for designing both CUIs and AUIs. Its primary use is for designing AUIs for cases that exclude the possibility to use an effective transformation from another input model. *UIP Visual Editor* supports a visual method for designing and modifying individual UIP application components: *AUIs*, *CUIs*, *Data Models* and *Event Handlers*.

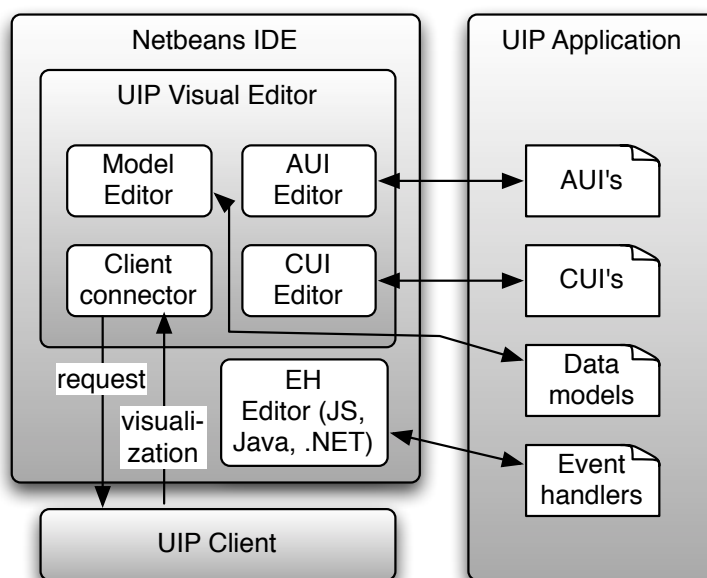


Figure 4.5: Components of UIP visual editor

Figure 4.5 shows the components of the *UIP Visual Editor*. It is implemented as a module for the NetBeans IDE [70]. *UIP Visual Editor* use *UIP application* structure (see 3.3.2) for data persistency of AUIs, CUIs, *Data Models*, and *Event Handlers*. As described above, UIP UIs are rendered on UIP clients using platform-native UI components. Implementation of consistent rendering of many heterogeneous platform elements embedded directly into the *UIP Visual Editor* would be complicated. Instead, this issue has been resolved by introducing a client-communication protocol that allows individual UI elements to be rendered on request directly to connected *UIP Clients* (or emulators). The resulting visualization is then transferred back to



the *UIP Visual Editor*.

The *UIP Visual Editor* has been developed as part of Master's Thesis [42] that was advised by this dissertation thesis author. More details about the *UIP Visual Editor* are described in section A.2 (appendix A).

### 4.3 Application audit and AUI transformation

This section focuses on AUI derivation directly from an application source code using its inspection. Many real-world applications are strongly based on data persistence and object-relational mapping – ORM [78]. Manual development of UIs for this class of applications leads to code-replication and is error-prone due to human errors [imp2]. Multiple target UI platforms make this problem even more serious.

In order to automate the process of AUI derivation, we apply the AspectFaces (AF) framework [imp2] to the audit application content and data-model. The main source of information used for basic UI structure is captured by the data models. This information is normally restated in the UI [15] or in models describing the UI. This impacts both development and maintenance efforts and can be a source of errors. To reduce the information restatement, or to even eliminate it, AF applies data model code-inspection through meta-programming and reflection mechanisms [36].

Many contemporary statically-typed programming languages have the ability to describe themselves, which is called Reflection [36]. This gives us an opportunity to inspect data classes, their fields and constraints. Such inspection considers constraints given by field annotations such as the Java Persistence API (JPA) standard [27], the validation standard [6], or presentation extension [17]. In addition, the inspection is open to definitions of new extensions captured by annotations, such as field visibility for a given geo-location, or user profile type, etc. The result of the inspection is a hierarchical structural model of the given data-model class.

An instance of this structural model, together with the system context, is the subject of aspect-oriented transformation. Details about the transformation process are described in [16]. It has four phases, see Figure 4.6. First, it adjusts the structural model instance accordingly to the runtime context. For example, the data field *'state'* is eliminated for a given user since he/she is from the Czech Republic using the AOP Annotation-driven participant pattern. Or a given field constraint is modified to be read-only for the given use in the UI. This phase updates the structural model instance.

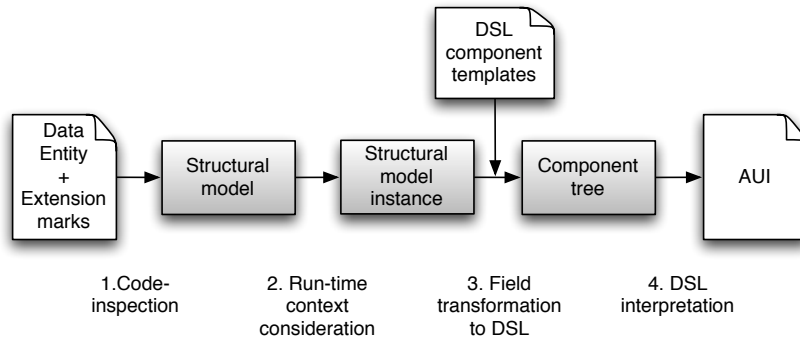


Figure 4.6: Code-inspection

The second phase aims to choose an appropriate DSL transformation template for each data field. Such template is selected by transformation rules. A transformation rule consists of a pointcut that forms the query on the structural model instance and context and gives advice on the DSL template. In our case, the DSL template contains the AUI specification for a given field type. The template does not refer to any specific data, but consists of target language constructs and integration constructs – composition rules [16].

The third phase occurs when a part of an UI that represents the given data is requested by a user. At this moment, all corresponding data fields are processed, resulting into generations of a component tree. Consider the following example of a data class with a text field. The text field has two annotations enforcing restrictions on the input: a JPA annotation restricting the text length, and validation annotation restricting email format. In this case a transformation rule for a text field is activated. On the basis of the pointcut we look for a match with a given property of the structural model instance and context: e.g. `type = 'text'` and `length < 100` and `type = 'email'` and `userRole  $\neq$  'visitor'`. This pointcut uses logical operations and combines `type`, `length`, and `email` from the structural model instance and `userRole` from the context.

Phase four is about interpretation of the DSL within the context and a given property of the structural model instance. In our case, the templates consist of the constructs of an AUI and of composition rules. Each composition rule consists of a pointcut and an advice. The pointcut uses exactly the same constructs as for the transformation, but the advice aims to integrate additional concerns to the AUI constructs. For example, in the case of a text-field, we ask whether there is a length restriction, and if so, we integrate a validation rule. Or we can look at the context specific part and say that when the user is an administrator, we allow him to submit

the field empty. Other users must nevertheless provide this type of information. The result of this phase is a an AUI. In this case, AF outputs AUI fragments that fully reflect the application data model and consider the application context (user roles, time, user-location, user device, user profile, etc.).

Next, we summarize the benefits: the correlation between the data model and AUI does not need to be developed, verified and maintained manually. This reduces the tedious work of UI developers and avoids human errors. Furthermore, consider that since AUI uses a language with no type-safety, it would be easy to cause an inconsistency by manual development, and so careful revision would be necessary. With the use of this automated process, all future changes to the data model are implicitly applied to the AUI fragments at runtime. Other benefits come from the AOP process characteristics – separation of concerns, support for systematic variations, parameterization, reduced code coupling and volume.

## 4.4 Transformation of specific input models

This section focuses on examples of transformation of selected domain-specific models into AUIs. Selected industrial standards for implementation of an intelligent household use specific internal structures to model controlling UIs. On an example of two different standards, we show how their internal model can be transformed into an AUI. The first standard discussed is Universal Remote Console (URC) [45]. The other example focuses on Home Automation Bus (openHAB).

### 4.4.1 Universal Remote Console Sockets

Universal Remote Console is anchored as International Standard ISO/IEC 24752 [45]. This standard specifies communications between a target device that a user wishes to operate, and a universal remote console (URC). *URC* presents the user with a remote UI through which they can discover, select, access and operate target devices. Each target to be operated exposes a *UI Socket* that defines controllable functions of the target device. Structure of *URC UI Sockets* is compatible to an *AUI* structure in the vocabulary of this work. *URC UI Sockets* are accompanied by *Resource Sheets* that contain other information necessary for building a UI, e.g. language constants, icons etc. *URC* uses central component to handle communication between targets and controllers – Universal Control HUB (UCH). This section describes basics about the *URC* integration, detailed information about this topic

can be found in [61].

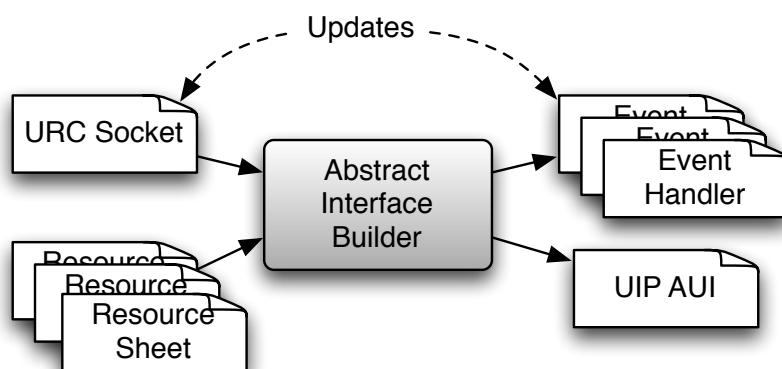


Figure 4.7: URC integration – derivation of AUI

Figure 4.7 shows a conceptual model of *URC Integration*. The *Abstract Interface Builder* is responsible for analysis of *URC* internal models, namely, *URC Sockets* and *Resource Sheets*. As a result *Abstract Interface Builder* generates AUIs and necessary part of the application logic in a form of *Event Handlers*. Their main purpose is to handle communication between *UIP Server* and *URC UCH* at runtime.

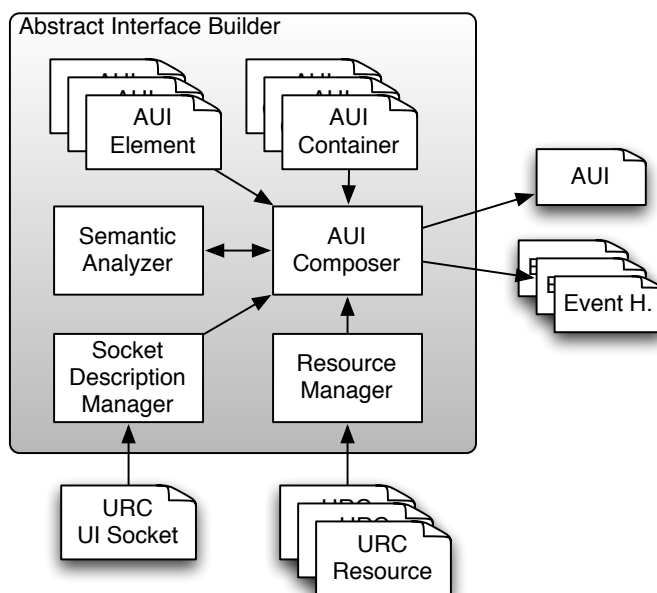


Figure 4.8: Abstract Interface Builder for *URC*

*Socket Description Manager* as depicted in Figure 4.8 is responsible for extracting relevant information from *URC UI Sockets*. Each *UI Socket* socket contains a set of socket elements and commands. There is also additional information about data

types of elements, sets of legal values, and restrictions expressing at what circumstances could be a particular value changed. Furthermore a *URC UI socket* contains a hierarchical structure of elements and their grouping into sets of related elements.

*Socket Description Manager* parses the XML description of given *UI Sockets* and provides the extracted relevant information to the *Abstract Interface Builder* in an object representation. *Resource Manager* parses additional information about *UI Socket* elements from the resource sheets provided by the *UCH API*. A *Resource sheet* has the form of an XML file, usually bound to a particular *UI Socket*. It contains information like description of the elements, labels for a particular language, icons etc. This information is later bound to the AUI and consequently to the FUI.

Purpose of the *Semantic Analyzer* is to determine some additional knowledge about the socket elements, in particular, if a resource sheet is unavailable. For example it recognizes media elements according the content of socket element value. For example, if a value contains .jpg,.png, or .gif, there is high a probability that the value refers to an image.

The earlier version of our *UIP Platform* was integrated with *URC* in the framework of the i2home project, for more details refer to [wos3] and [scopus4].

#### 4.4.2 OpenHAB

The *Open Home Automation Bus* (openHAB) [99] project aims at providing a universal integration platform for home automation. Similarly to *URC*, *OpenHAB* exposes control UIs in a form that can be translated into an AUI. In the vocabulary of *OpenHAB*, this structure is called *Sitemap*. Unlike *URC Sockets*, *Sitemap* is a com-

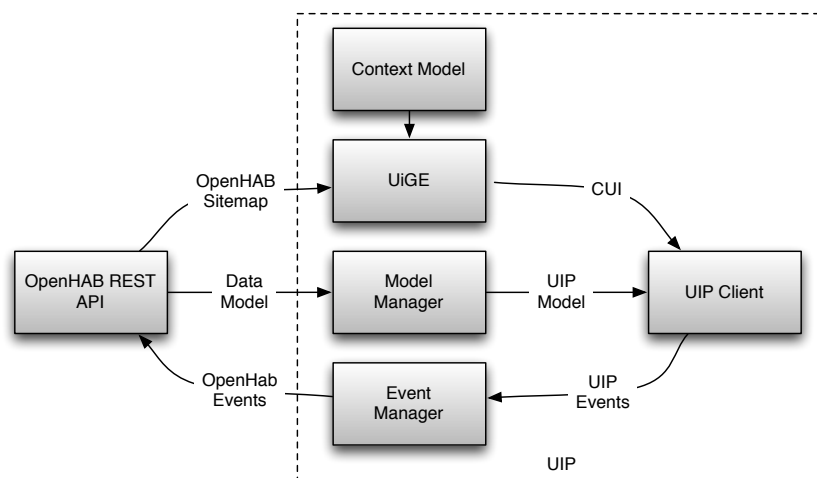


Figure 4.9: Integration of OpenHAB

posite structure for all devices and services integrated with the current *OpenHAB* system.

Figure 4.9 illustrates integration of *OpenHAB* and the *UIP Platform*. An *OpenHAB Sitemap* is translated into an AUI in a similar manner to the translation of a *URC Socket* mentioned above. This AUI is later transformed into the context-sensitive CUI using *UiGE* (see chapter 6). Relevant data are exposed to *UIP Clients* and *Event Handlers* using the connection of *Model Manager*. This connection can be implemented using special persistent *Event Handler*. In the opposite direction, relevant *UIP Events* are translated into the *OpenHAB events* and propagated into current *OpenHAB* runtime.

## 4.5 Conclusion and Contribution

This chapter described multiple methods to derive AUIs – an immediate input for automatic CUI generation. Various methods were discussed, which shows the versatility of our approach from the input perspective. In general, derivation of AUI further simplifies implementation of context-sensitive UIs. Furthermore, capabilities of platforms that support model transformation into *UIP AUIs* can be effectively extended by instant support of multiple heterogeneous UI platforms proving context-sensitive UIs.

# Chapter 5

## Context Model

This chapter focuses on *Context model* that corresponds to the requirements of *CUI* generation described in this thesis. Figure 5.1 depicts how *Context model* relates to the general UI generation pipeline. Most importantly, the *Context model* attributes the *CUI generation*. However, the *Context model* is accessible from other components of the *UIP Platform*, e.g. to enable manual UI adaptations. Additional information about *Context model* and its evaluation can be found in [wos4].

### 5.1 UIP Context Model

This section focuses on the *UIP Context Model – UIP CM*. *UIP CM* describes internal and external factors that affect an appearance of the resulting *CUI* and its elements. It consists of four sub-models: the DM, the UM, the EM and the Assistive Technologies Model (ATM), as depicted in Figure 5.2. The properties in the context model have a direct relationship with the properties of the UI elements in resulting *CUIs*. Property values in UM, DM and EM descriptions are independent. In the event that any sub-model is missing, the *UIP CM* still provides a useable default.

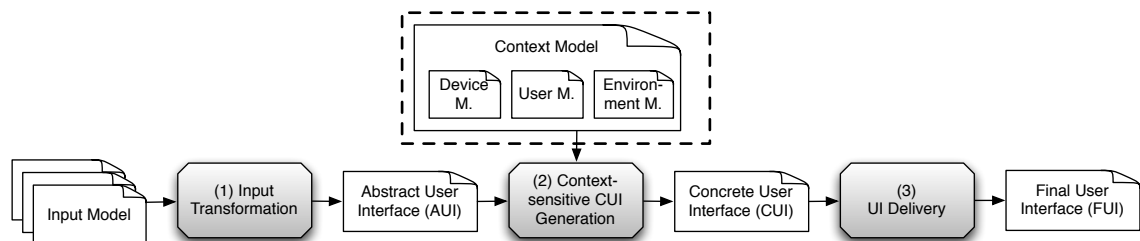


Figure 5.1: Scope of context model in the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

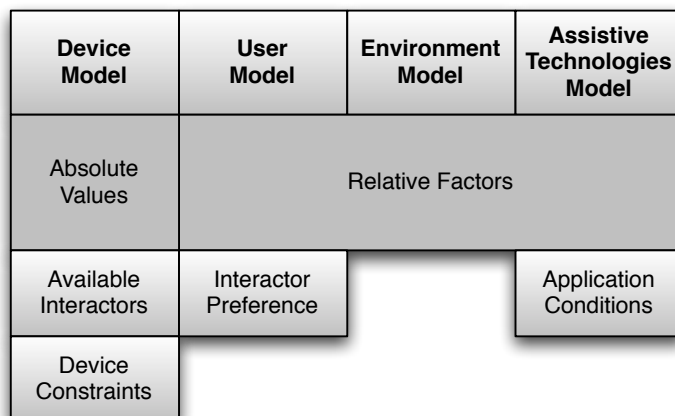


Figure 5.2: Components of Context Model

In contrast to other approaches for context modeling, the *UIP CM* focuses is on individual properties of UIs, such as font-size, element-spacing, sound-volume, etc., which are reflected in individual sub-models.

### 5.1.1 Device Model

*DM* reflects attributes of actual device that can affect the interactive sessions. Unlike other sub-models, the *DM* contains device-specific default values of UI properties in absolute values (e.g. 10 pixels).

Additionally, *DM* contains the set of available CUI elements (in some vocabularies called interactions) that can visualize information or provide users' input. *DM* also contains information about device constraints, such as screen width and screen height, maximum volume, maximum contrast, etc. Example of *DM* instance is in appendix B, section B.2.1.

### 5.1.2 User Model

Unlike the *DM*, all other sub-models, including the *UM* contain relative factors rather than absolute values of UI properties. Example in Figure 5.3 explains this concept. Furthermore, the *UM* contains information about user-specific interactor preference. This information is used as a guideline during the automatic CUI generation in order to meet users' preferences. Example of *UM* instance is in appendix B, section B.2.2.

In accordance with the ability based design [113], *UM* describes user abilities as their effect on the appearance and usage of individual UI components. Some



limitations the restrict interaction can be temporary [other4] or even caused by user fatigue [scopus5].

### 5.1.3 Environment Model

*EM* reflects the effects of the current environment on an interactive session. Use of context sensors (see 5.2) is expected for automated actualization of the *EM*.

### 5.1.4 Assistive Technology Model

*ATM* describes the effects of assistive technologies (assistive aids) to the properties of CUI elements. E.g. use of prescribe-glasses enables the user to see smaller fonts. On the contrary, protective gloves require significantly bigger touchable areas in touch-based UI or prevent touch-based interaction at all.

### 5.1.5 Computation of final context model property values

The final context-dependent value of a particular UI property can be computed as the product of properties in individual sub-models. There are absolute values in DM and relative factors in the other sub-models.

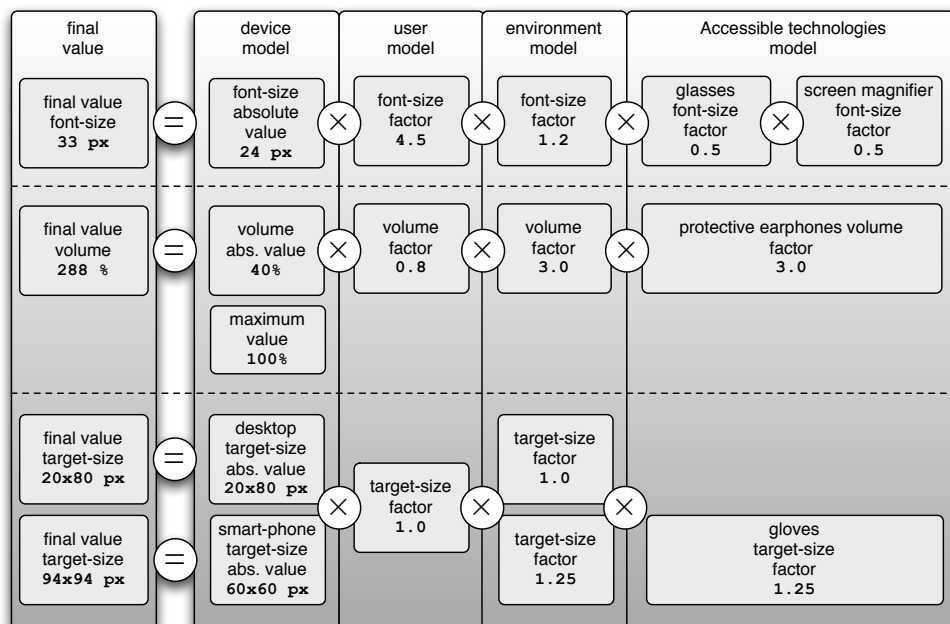


Figure 5.3: Computations of final values of context-sensitive UI properties, example computation of font-size property (top), volume property (middle) and target-size property (bottom)

In the top part, the Figure 5.3 depicts a demonstrative computation of the font-size property. In *DM*, there is an absolute value of 24 pixels. This is a default value that fits the needs of able-bodied persons when using the device in an ideal environment. The value is defined as an average value measured on able-bodied users. The *UM* contains a relative factor of 4.5, meaning that the user needs all fonts to be 4.5 times bigger than the average. In *EM*, there is a font-size factor of 1.2 (bright ambient light detected by a device sensor). There are two assistive technologies involved in the interaction: glasses and a screen magnifier. Both of them have a relative font-size factor of 0.5. The total font size is computed as the product of all relative factors in *UM*, *EM* and *ATM* and the absolute value in the *DM*. In this case, the final value is 33 pixels.

Figure 5.3 – bottom depicts computation of the target-size value. Target-size represents the dimensions of active areas such as buttons that are triggered by mouse clicks or finger taps. This example shows the differences in the final value for two different devices – a desktop and a smartphone. The default values in *DMs* are different, which affects the final value. In the case of the smartphone, a constant device movement has been detected. The precision of the tapping is therefore worsened. This is reflected as a relative factor of 1.5 in *EM*. The user’s pointing and tapping ability is normal, so the relative factor in *DM* is 1.0 (no-effect). In addition, the user wears gloves because of the cold environment, and this further decreases the precision of the tapping. Consequently, for the desktop the default value in *DM* is used (all factors in the context model are equal to 1.0). In the case of a smartphone, the final value is  $94 \times 94$ px.

The context model also contains properties that express the users’ cognitive ability to interact with information in a specific domain. These properties are expressed as a factor from 0.0 to 1.0, where 0.0 expresses that the user is not able to understand any information in the specific domain. Factor 1.0 expresses that the user is able to understand all information in the specific domain. Correspondingly, individual elements in the AUI can have an information complexity property. This property indicates the minimal value of the users’ domain specific cognitive ability property required for understanding information expressed by a particular AUI element. During the CUI generation process, elements that express information which is excessively complex and are not required (the importance property of the AUI element is not equal to 1) are omitted from the CUI generation process. The intermediate values on this scale (e.g. 0.7) cause omitting of complex UI elements and functionality typical for expert users. The values of the information complexity property

and the domain specific cognitive ability property are currently defined manually by experts in a given domain. Cognitive ability property can be set manually by user to reveal more complex functionality.

## 5.2 Context sensors

Changes in the CM are detected during the application runtime and propagated into the CUI generator. If a change occurs, e.g. the user stops wearing gloves, the new CUI is pushed into the client device and rendered. The current position in the UI (e.g. the active field of a form) is stored in a special data model and it is preserved after the CUI update. In this way, the possible negative effect of the transition between two UI representations is reduced. Some changes of context model can be directly or indirectly detected by device sensors. For example, the intensity of the ambient light can be detected by smartphone sensors. In an experimental setup, we used a computer-vision method to identify a user (assign the corresponding *UM*) and to detect whether he/she is wearing glasses (in order to assign a corresponding *ATM*). The context model was adapted to the situation, and the update resulted in an update of the currently displayed CUI.

## 5.3 Conclusion and Contribution

In this chapter, we introduced *UIP CM*, a context-modeling method that can be integrated with other components of the *UIP Platform*, most importantly with the *UiGE*. In comparison to current approaches listed in section 2.4, following benefits of our solution can be noted:

- *Novel CM design with independent factors in particular sub-models.* This feature simplified both CM development as well as its later use. Each CM component can be developed and maintained separately.
- *Our CM focuses on effect to resulting CUI.* Unlike other approaches for the context modeling, our approach model context-factors as their effect on the result. Many current CMs contain complex sets of various attributes of the user or environment without direct relationship to UI properties.
- *Our CM provides useable default values when any sub-model is missing.* Except the DM, using the concept of relative factors, the CM provides useable default

if any sub-model is not available.

- *Less complex than models based on ontologies.* Many approaches for context modeling bases on ontologies, see section 2.4. However, these approaches can express complex situations and automatic reasoning is possible, in real-world situations, they can became really complex. Consequently useable ontology-based context models are usually hard to develop and maintain. Furthermore, a complex ontology-based CM can be transformed into our context model, however, this is subject of the future work.

In the previous chapters, *UIP Platform* components necessary for context-sensitive UI generation have been described. The following chapter focuses on the main topic of this thesis – UIP User Interface Generator (UiGE).

# Chapter 6

## User Interface Generation and Optimization

This chapter focuses on the automatic CUI generation. Figure 6.1 shows this phase in the framework of general UI generation pipeline. In section 6.1, we focus on the definition of *optimization function*. Definition of basic optimization function is followed by description of more sophisticated cases based on its parameterization. In the second part of this chapter (starting with section 6.2), we focus on the realization of CUI generation process in the framework of the *UIP Platform*.

### 6.1 CUI optimization

In the framework of this thesis, CUI generation is defined as an optimization problem. It is therefore necessary to define optimization metrics that assess optimality of generated UIs in order to automatically compose optimal results. Currently, the UIs are typically evaluated using usability testing with target user audience or by expert evaluation carried out by human experts based on heuristics. However, for purposes

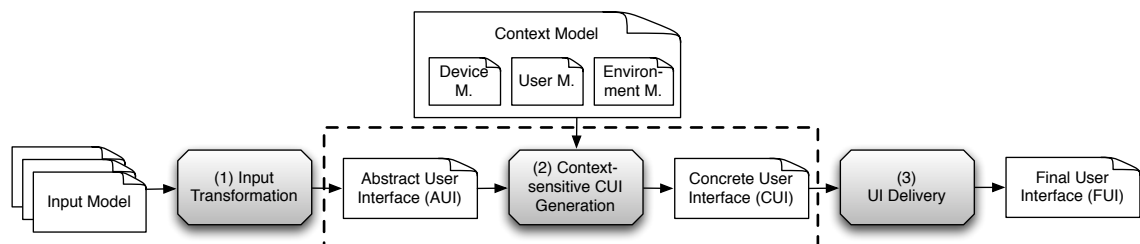


Figure 6.1: UIP User Interface Generator in the scope of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

of automatic CUI generation, we need an automatic assessment of UI quality. The current context of use should also be considered.

The aim of the UIP User Interface Generator (UiGE) design is to provide general solution for transformation of AUIs into CUIs. This transformation should reflect the usage context and support a wide range of heterogeneous target UI platforms. In order to achieve such a requirement, the UiGE uses simple basic optimization function that can be later parametrized by more complex attributes (in our vocabulary called optimization heuristics, see section 6.1.1). The UiGE also communicates with the target UI device during the process of CUI generation. This method in accordance with the simple basic optimization function enables UiGE to support hererogenous target UI platforms. The UiGE core does not need to be modified in order to support a new target UI platform.

Statement 6.1 defines basic the optimization function for *UI Elements*. The value expresses cost that corresponds to user effort estimation. The lower the cost value is, the lower interaction effort is required from the user.

$$cost(steps, h) = (1 + steps) \times \prod_{i=1}^n h_e(i) \quad (6.1)$$

where *steps* corresponds to number of integration steps (e.g. mouse click on certain position) needed to reach the possibility of immediate value selection,  $h_e(i)$  refers to a value of an applicable heuristic,  $n$  is the number of all heuristics that are applicable to a particular mapping. For non-interactive elements the resulting *cost* = 1 in case there are no applicable optimization heuristics.

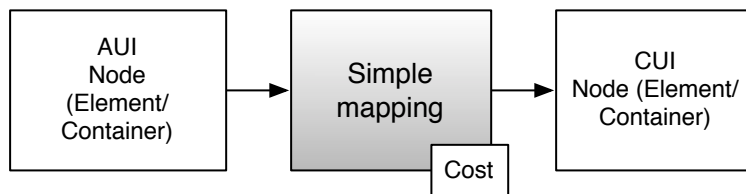


Figure 6.2: Simple mapping

Figure 6.2 shows *Simple Mapping* that determines the relationship between an AUI Node (*AUI Element or Container*) and an CUI Node (*CUI Element or Container*). Typically, one *AUI Node* can be mapped to multiple different *CUI Nodes*. Each mapping has a *cost* value that for *Elements* corresponds to the optimization function defined above.

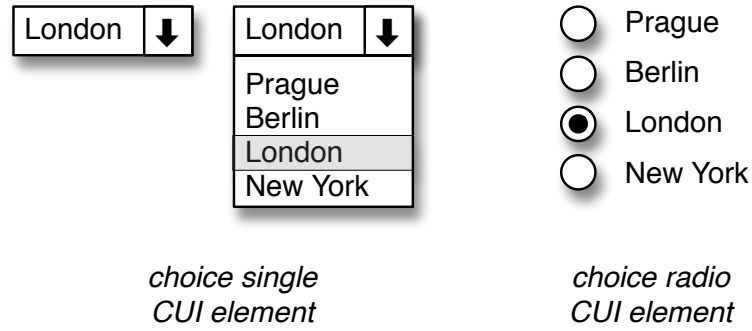


Figure 6.3: Example of simple mapping

The following example illustrates the simple mapping. Let us have an input AUI element with a restriction property specifying that possible values can be "Prague", "Berlin", "London", or "New York". Figure 6.3 shows examples of CUI elements that can be mapped to this type of AUI element. The first example is mapping to *choice single CUI element* (on most UIP clients represented by combo-box, see left part of Figure 6.3), let us call it *combo-box mapping*. The second example is a similar mapping to *choice radio CUI element* (on most UIP client represented as set of radio buttons, see right part of Figure 6.3), let us call it *radio set mapping*.

In the case of *combo-box mapping*, the value of the basic cost function equals 2 as user needs one additional step to be able to choose a particular value (e.g. Berlin). He or she must expand the combo-box element to be able to select a desired value. In case of *radio set mapping*, a desired value can be selected immediately (cost = 1).

Often, mappings to elements with lower values of *cost* result in CUI elements that are more space-consuming. In nontrivial cases this leads to the necessity to divide resulting CUI into more screens (e.g. tabs container) or to use scrolling that also makes the interaction more complex. Therefore also *AUI to CUI container* mapping has corresponding cost value.

Statement 6.1 shows computation of optimization function for containers:

$$cost_c(steps, cost_n) = (1 + steps) \times \prod_{i=1}^n h_c(i) \times \sum_{i=1}^n cost_n(i) \quad (6.2)$$

where *steps* value corresponds to the estimated average number of integration steps needed to reveal internal nodes wrapped into a particular *Container*,  $h_c(i)$  refers to a value of a heuristic applicable a particular *Container*.  $cost_n(i)$  is cost value of particular *Node* (*Element* or *Container*) wrapped into a particular *Container*.

### 6.1.1 Optimisation heuristics

Basic optimization function reflects only one a simple metric – the number of interaction steps related a particular UI element. Optimization heuristics enable to enhance the optimization function with more complex parameters. Figure 6.4 illustrates the application of a *heuristics rule* to the cost function of a mapping. During the application process, complex *heuristic rules* can also access the *CM*. Similarly to CM value computation, a heuristic rule should affect the cost value by a multiplication factor. A single heuristic rule is applied only once to each mapping (Simple or Template).

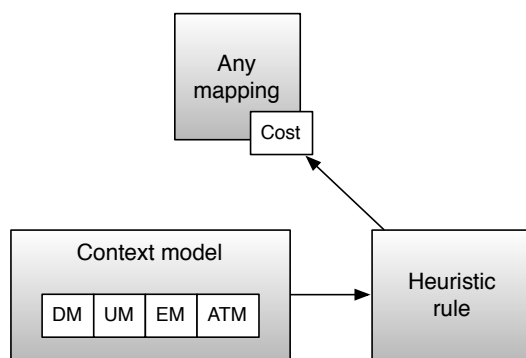


Figure 6.4: Heuristics rule

In the following text, we will show an example how a heuristic rule can be used to select better representation for elements with higher importance. Let us have two AUI elements (A and B) that represent one of  $N$  selection ( $N = 3$ ) and two possible CUI representations *combo-box mapping* and *radio set mapping*. The situation is illustrated in Figure 6.5. Let us assume, that *combo-box mapping* has  $size = 1$  in the resulting CUI, whereas *radio set mapping* has  $size = 3$ . In the basic case, the cost value of the *combo-box mapping* will be  $cost = 2$  and for the *radio set mapping*  $cost = 1$ . The task is to find the optimal combination in case we have to put all representation into resulting CUI interface (we assume that there is no spacing between CUI elements).

In the case there is enough space – CUI interface  $size \geq 6$ , the optimal representation is to use *radio set mapping* for both AUI elements (variant 0 in Figure 6.5). In this case the total  $cost = 2$ . The minimum CUI interface space is used when we use a *combo-box mapping* for both AUI elements, in this case the total  $size = 2$  (Figure 6.5 variant 3). In this case, the total value of  $cost = 4$ . In case that the size is between  $2 < size < 6$ , it is possible to use *combo-box mapping* either for element



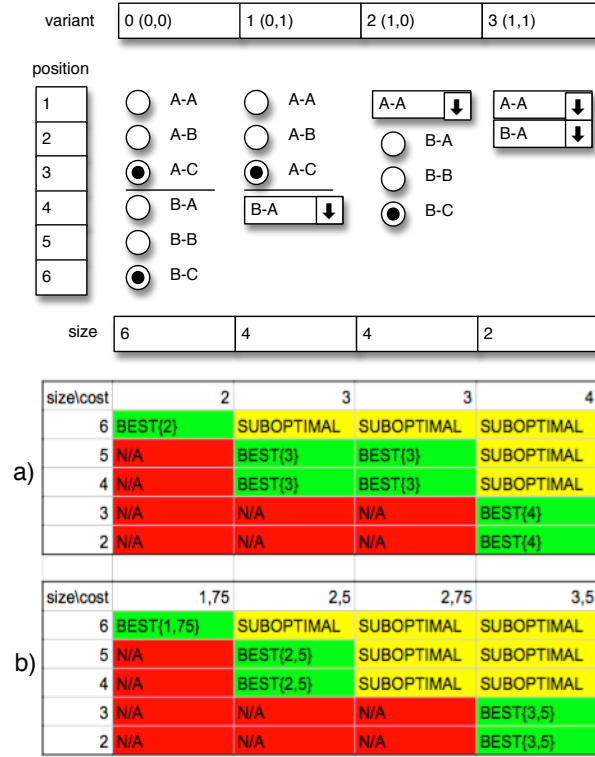


Figure 6.5: Example of importance heuristic.

- a) Shows the situation that there are no applicable heuristics (the cost value corresponds just to the number of integration steps).
- b) Shows the situation with an importance heuristic applied to cost values of individual mappings.

A and *radio set mapping* for element B or vice-versa. In both cases, the resulting total *cost* = 3 (variants 1 and 2).

Let us assume, we want to reflect the importance of elements. Importance is a value between 0 and 1. The most important element has the *importance* = 1. In our case, an importance value is already defined in the AUI. AUI element A has the *importance* = 1 and element B has the *importance* = 0.75. The statement 6.3 defines our heuristic rule to reflect element importance.

$$h_e(\text{importance}) = \text{importance} \quad (6.3)$$

In this case, each cost value of relevant mapping is multiplied by the heuristic value, in our case directly with the importance value. In our case it means that less important elements will also have lesser value of the cost function. Consequently, mappings with higher basic cost values (number of interaction steps) will be used for less important elements. The situation is illustrated in Figure 6.5 b). We can

see that variant 1 that prefers better representation for AUI element  $A$  is used when the available space is  $2 < size < 6$ .

### 6.1.2 Templates

The above described *Simple Mapping* provides basic AUI to CUI transformation optimized accordingly to rather simple optimization metrics. In some cases, we need to configure the transformation process to provide exact results. In the Figure 6.6, we can notice that the *Template mapping* provides a transformation of more complex parts of an AUI into more complex CUI structures.

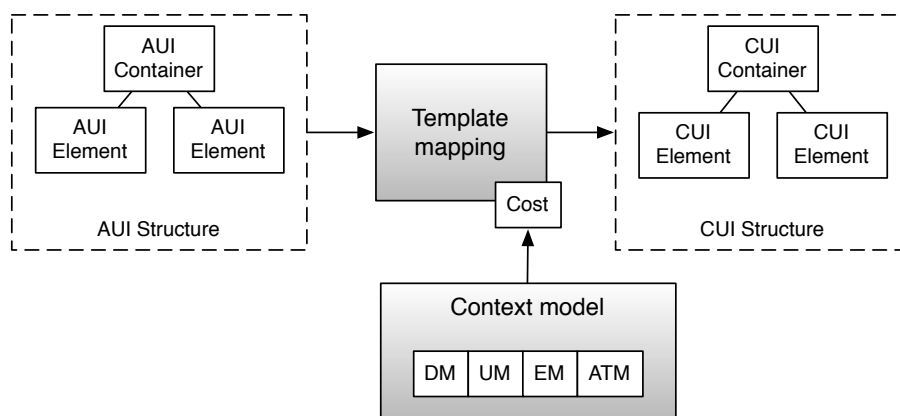


Figure 6.6: Template mapping

## 6.2 CUI Generation Process

This section describes the realization of context-sensitive CUI generation in the framework of *UIP Platform*. Simplified scheme of the CUI generation process is depicted in Figure 6.7. The resulting CUI reflects the current context and can be directly rendered on a particular platform.

The input to the automatic CUI generation process is an AUI, which structure is described in section 4.1. The goal of the UI generation process is to generate UIs that correspond to optimization metrics. Primary optimization metric is the *Basic optimization function* as defined in section 6.1 for UI Elements and Containers. As described above, the relationship between AUI nodes and components of the CUI is specified by the mappings. Each mapping provides corresponding cost value that is used for optimization during the CUI generation.

The *AUI* hierarchy also includes *AUI Containers* (internal nodes of the *AUI* tree) that divide the elements into a hierarchy of groups. The *UiGE CUI generator* reflects this structure to visualize element grouping. In some cases, *UiGE* uses the *AUI* hierarchy to render resulting *CUIs* using a multi-screen layout, if necessary (e.g. using tabs or wizard *UI*).

During this process, the *UiGE CUI generator* communicates with the corresponding *UIP Client* in order to get information about the final element representation on a particular platform (most importantly its dimensions). Using this approach, the mapping definitions can be independent of the target client platform. Dynamic changes in both Context Model and *AUIs* are reflected by the *CUI generator* in runtime, and updated *CUIs* are propagated to relevant *UIP clients*.

Next, we provide a description of the *UI generation process* (see Figure 6.7). At the beginning, a set of all possible mappings and templates is reduced according to the current context model to a set of feasible mappings and templates. Both mappings and templates have their particular cost function value that corresponds to the estimated user effort while interacting with a particular *UI* component.

In the next phase, mappings and templates are ordered according their cost value. This is followed by the optimization process, which finds the optimal mapping, based on the cost-function values for individual *AUI* elements with respect to the context model constraints. Our optimization process uses a branch and bound algorithm [68] to find an optimal solution. It provides a solution within a few seconds for typical instances. The product of the *UI generation process* is a *CUI* that is immediately transferred to the particular *UIP Client* and consequently displayed.

Figure 6.8 depicts the effect of the so-called *Small model update*. In this case, the resulting *CUI* can seamlessly render *UI* properties that were updated as a result of

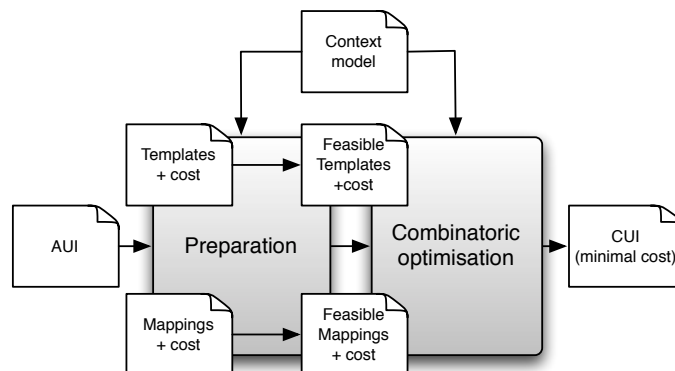


Figure 6.7: Simplified scheme of the *UI generation process*

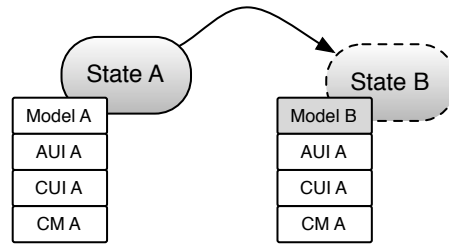


Figure 6.8: Small model update

a *Model Update*. The update is performed on client-side using model-wide binding (see section 3.2.1).

If a *UIP Client* is not able to reflect any *Model Update*, it propagates special *Event* to the *UIP Server*. This event is then routed to the *UiGE*. As a result, *UiGE* re-generates affected part of the resulting *CUI* structure. This action is illustrated in Figure 6.9.

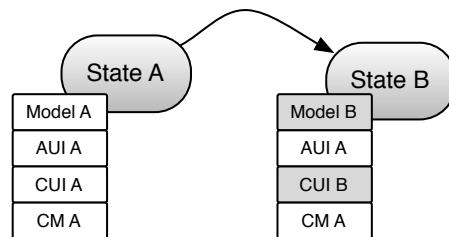


Figure 6.9: Substantial model update

Figure 6.10 illustrates the situation, when an input AUI changes. Such a situation is typically triggered by change of another high-level input model. In such a case, the AUI update triggers the CUI generation process which results into a corresponding update of resulting CUI.

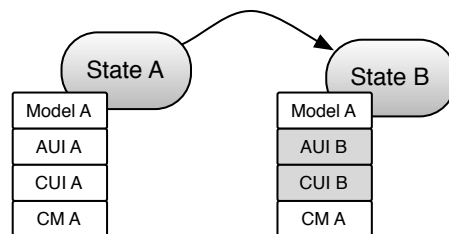


Figure 6.10: AUI update

Finally, Figure 6.11 illustrates situation when *Context Models* changes. If the change is substantial, it triggers CUI generation process. The resulting CUI then corresponds to new conditions reflected in the updated CM.

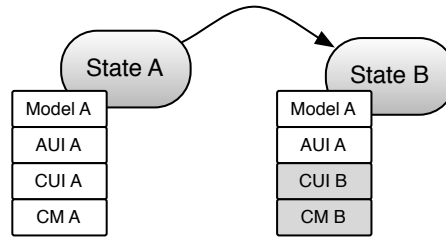


Figure 6.11: Context model update

### 6.3 Conclusion and Contribution

In this chapter, we described our method for automatic CUI generation. In the first part, we focused on the definition of a corresponding optimization metric. Our optimization metric reflects the number of steps that need to be carried out to perform typical operations with resulting UIs. For more complex cases we have defined *optimization heuristics*. Using this method the optimization process can be configured for specific application domains while preserving its generality. The relationship between AUI and CUI elements is defined by mapping. Apart from the simple mapping and the container mapping, we have also defined a template mapping. This type of mapping can be used to configure the UI generation process to provide predictable results for specified cases while preserving overall generality of the process.

In the second part of this chapter, we focused on the realization to the UI generation process. The aim of this process is to find the minimal total value of the optimization function while reflecting the current instance of the Context Model. UIP User Interface Generator (UiGE) communicates with the target *UIP Client* to determine final dimensions of UI elements and layout. Using this method, we can support all compatible clients while preserving general realization of the UiGE.



# Chapter 7

## User Interfaces based on application source-code audit

Previous chapters focused on the foundations and the methodology and basic principles behind the Context-sensitive Automatic Concrete User Interface Generation Pipeline and its implementation – the *UIP Platform*. Subsequently, this chapter and following three chapters focus on various applications of the *UIP Platform*. The aim is to show in greater detail methods described in this dissertation thesis. Here we use real-world applications rather than solely examples based on the theoretical basis. Furthermore, a variety of the applications based on *UIP* illustrate the flexibility of the approach described in this thesis.

At first, this chapter focuses on a use-case of the *UIP Platform*, where a source-code analysis method is used to derive AUIs – an immediate input for the UIP User Interface Generator. This example illustrates the capabilities of our approach well as the UiGE Pipeline is used to the full extent.

Many applications, mostly business oriented, are based on the data persistence and on the object-relational mapping – ORM [78]. The manual development of

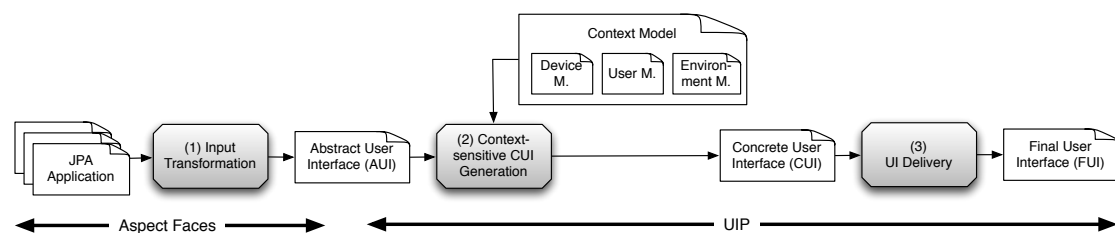


Figure 7.1: Data-driven applications in the scope of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

UIs for this class of applications leads to code-replication and is error-prone due to human errors [imp2]. Support for multiple target UI platforms at for a single application makes this problem even more serious. We suggest a solution where the input is extracted from the application backend using a code-inspection method. On an example, we show how to automatically generate context-sensitive UIs for various target platforms using the code-inspection.

Figure 7.1 shows the UiGE Pipeline and the related scope of the code-inspection method described in this chapter. Firstly, AUIs are derived using the *Aspect Faces* engine. Extraction of data necessary for this type UI generation has been already described in chapter 4, section 4.3, In the next phase, *UIP Platform* provides automatic context-sensitive generation of UIs for various platforms. In this case, the resulting UIs are represented as forms.

## 7.1 Application source code audit

In this section, we show on a practical example how the *Aspect Faces* framework uses the code-inspection (application source code audit) to derive an AUI. This AUI is in the next phase used as an input for context-sensitive automatic CUI generation.

Listing 7.1 shows an example of source-code of an application that uses ORM for data-persistence. Note annotations in the source code (introduced by "@").

```
@Table(name = "person_info")
public class PersonInfo extends EntityObject {

    /*fields*/

    @Column(name = "first_name", nullable = false)
    @Length(max=100) @NotNull
    @Pattern(regexp="^[^\\s].*")
    public String getFirstName() {return this.firstName;}

    @Column(name = "last_name")
    @Length(max=100) @NotNull
    @Pattern(regexp="^[^\\s].*")
    public String getLastName() {return this.lastName;}

    @Email @NotNull
    @Length(max=255)
    public String getEmail() {return this.email;}
}
```

Listing 7.1: Example data entity used for generation of forms in Figures 7.2 and 7.3



## 7.2 Resulting UIs

In the following text, we provide two proof-of-concept examples of UIs generated using the UIP Platform when the application source code audit is used as an input.

The first proof-of-concept example focuses on a case of a population census form. Its aim is to show the capabilities of our approach to generate context-sensitive, platform-aware electronic equivalents of real physical forms. This example was derived from a selected form of census in the Czech Republic in 2011. A set of forms for three different platforms was generated from a *single AUI*. The AUI was generated through AspectFaces code-inspection of the underlying JPA application backend.

Figure 7.2-b shows a visualization of a UI generated for a desktop platform. Font-size, element-size, element spacing and layout are influenced by the context-model. Using model-wide binding and server-side application logic, the UIP client displays warnings next to elements with content that does not pass the validation criteria. The most suitable mapping to actual CUI elements that visualize individual AUI elements is determined using combinatoric optimization.

The figure displays two versions of a census form. The left version (a) is for an iPad tablet, showing a compact layout with small text and input fields. The right version (b) is for a desktop PC, showing a larger layout with larger text and input fields, and includes radio buttons for gender and country selection.

**Tablet UI (a) Fields:**

- Name and surname: First name, Surname
- Birthplace and gender: Personal ident. no., Date of birth, Gender
- Country: Country, Other citizenship
- Place of residence: District, Municipality, House no., Part of municipality, Orientation no., Street, In another state, Country
- Marital status: Marital status
- Registered partnership: Registered partnership

**Desktop UI (b) Fields:**

- Name and surname: First name, Surname
- Birthplace and gender: Personal ident. no., Date of birth, Gender (female, male)
- Country: Country (Czech, Other, without country of citizenship), Other citizenship
- Place of residence: District, Municipality, House no., Part of municipality, Orientation no., Street

Figure 7.2: UIs generated for: a – iPad tablet (left) and b – desktop PC (right), UI for iPad is generated using templates, UI for desktop PC is generated without templates

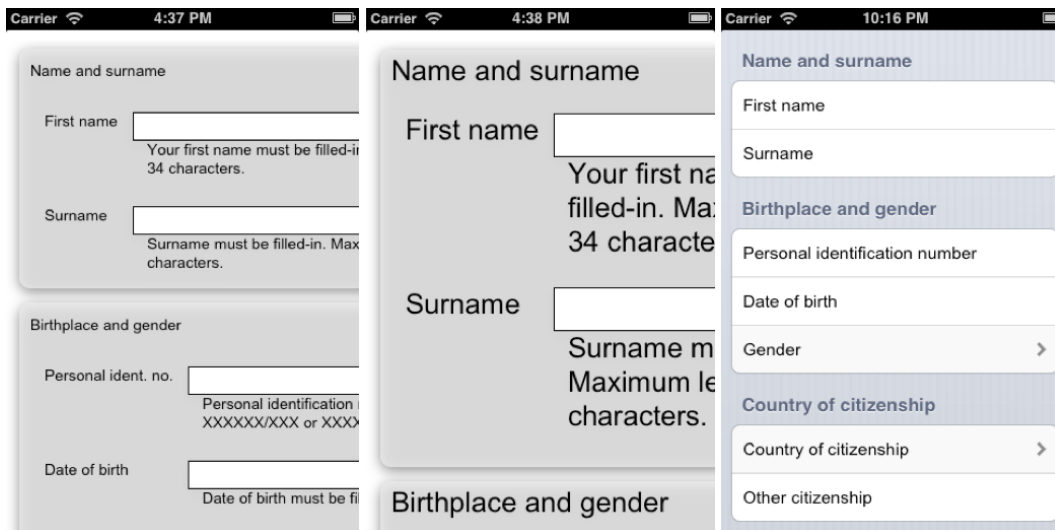


Figure 7.3: UI generated for iPhone: a – default context (left), b – for user with lower vision (middle), c – generated using templates (right)

Figure 7.3 shows the different variants of UIs generated for an iPhone UIP client. Figure 7.3-a shows a UI generated without templates with the default context model. Figure 7.3-b depicts a UI that is generated for a user with slightly reduced vision and with dexterity problems. Note that the size of the labels, and also the size of the interactive elements, is bigger. Figure 7.3-c shows an iPhone UIP client UI that is rendered using a complex native structure – UITableView. Because UITableView is a platform-specific structure, it was necessary to use the UIP template in order to map part of the AUI to such a structure. Figure 7.2-a shows a UI generated for an iPad UIP client. As in the case of the iPhone client, a template was used to generate this UI.

The UIs generated using this approach respect the usage context constraints. An example of such an interface generated for the iOS operating system is presented in Figure 7.3-a. In some cases, it is favorable to use specific platform elements. An example of such a component is an iOS table (UITableView). This structure provides a very good user experience, but it is a specific component of the iOS platform. It can be mapped only when the AUI contains a specific structure (a subset of AUI). The implementation of a mapping that provides the relationship between the AUI elements and such specific structures will make the UiGE generator too complicated and too hard to maintain. In order to address this issue, the UiGE is extended to support UI templates. A UI template is a complex mapping variant that provides the relationship between a subset of an AUI and a platform-specific structure. An example of a UI generated using a template is presented in Figure 7.3-c.

## 7.3 Conclusions and Contribution

In this chapter we have presented an example how the application source-code audit can be used to generate context-aware UIs at runtime. For data-oriented applications, our platform is based on effective code inspection and an aspect-oriented approach. The proposed method significantly reduces the amount of replicated code, which simplifies both development and maintenance (see section 10.3). Various client platforms with various capabilities are supported. In contrast to comparable approaches, UI elements native on individual platforms are used. For more details, please refer to [scopus2].



# Chapter 8

## Application: Indoor navigation for users with limited navigation and orientation abilities

This chapter focuses on an adaptive indoor navigation system for large indoor environments. Namely, we focus on a navigation system tailored to the hospital environment and individuals with limited navigation and orientation capabilities. Our study [other5] shows navigation problems the senior population has to deal with. Design of this in-hospital navigation system reflects our experience with this specific user group, see [wos2]. Initial version of this in-hospital navigation system was described in [scopus1]. Publication [other3] focuses on details of software realization of the in-hospital navigation system.

Figure 8.1 highlights stages of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline) in the relationship to basic components of the below described hospital navigation system. From the perspective

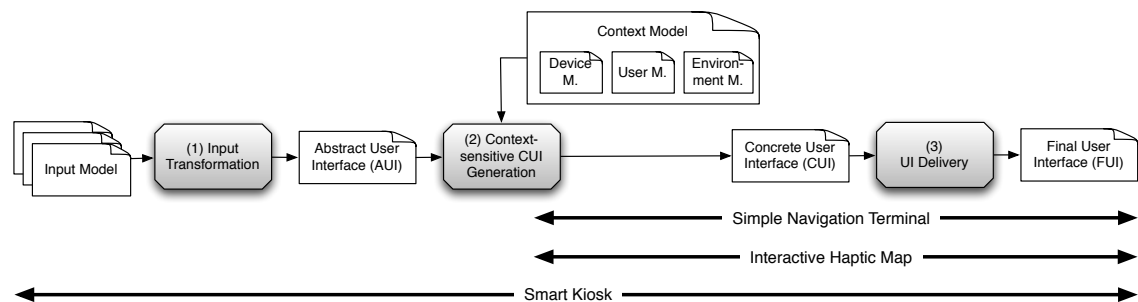


Figure 8.1: In-hospital navigation system components in the scope of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

of the UiGE Pipeline, the *Smart Kiosk* represents a navigation system component that employs the automatic CUI generation. On the contrary, less complex components like the *Simple Navigation Terminal* or the *Interactive Haptic Map* mostly employ only the CUI delivery stage of the UiGE Pipeline as marked in the Figure 8.1. However, even those components offer basic context-driven adaptations.

In the recent decades, the use of electronic outdoor navigation systems had spread significantly. The most important factors that have enabled this growth are the availability of compact electronic devices and the public accessibility of Global Navigation Satellite Systems (GNSS), such as GPS and Glonass [57]. By contrast, the development of electronic indoor navigation systems has been considerably slower. The main reasons have been the lack of a widely-used, globally available and reliable positioning system like GNSS for use in the indoor environment, and the need to develop standardized plans (or maps) of indoor environments.

Most present-day electronic indoor navigation systems require their users to carry a special single-purpose electronic device in order to use them. Many systems also require special equipment to be installed in the indoor environment, e.g. sensors or various kinds of ultrasonic, infrared or Bluetooth navigation beacons.

An inappropriate hospital navigation system can cause difficulties for the majority of visitors, but also for the personnel. Study [26] carried out in a hospital in the United States showed, that personnel at a 300-bed hospital spends about 4,500 hours annually to assist patients and relatives who cannot find their way.

According to [41], assistive technologies that require users to carry and operate special equipment that is recognized by the navigation systems place an extra burden on users with a disability. These individuals would benefit from lightweight discrete aids that incorporate devices that are popular among the general public, e.g. mobile phones. Many current ICT solutions struggle to fit people with disabilities to standard systems, using various assistive technologies. Unlike most present-day electronic indoor navigation systems, our solution does not rely on any particular device that the user is required to carry.

A computer-aided hospital navigation system for users with special needs require a specific infrastructure. This chapter shows how is the *UIP* platform used to deal with such requirements.

## 8.1 Navigation system design

Our navigation system does not require users to carry any physical object (an electronic device or even an identification card) to be able to use the system. Instead, the system uses face recognition to identify a particular user, and provides him/her with personalized navigation instructions to proceed to the next navigation point, or to the destination. In addition, the system targets a user audience with limited navigation and orientation abilities, e.g. visually-impaired persons and seniors. In order to meet these goals, the navigation system employs various types of adaptive navigation terminals that will be described below.

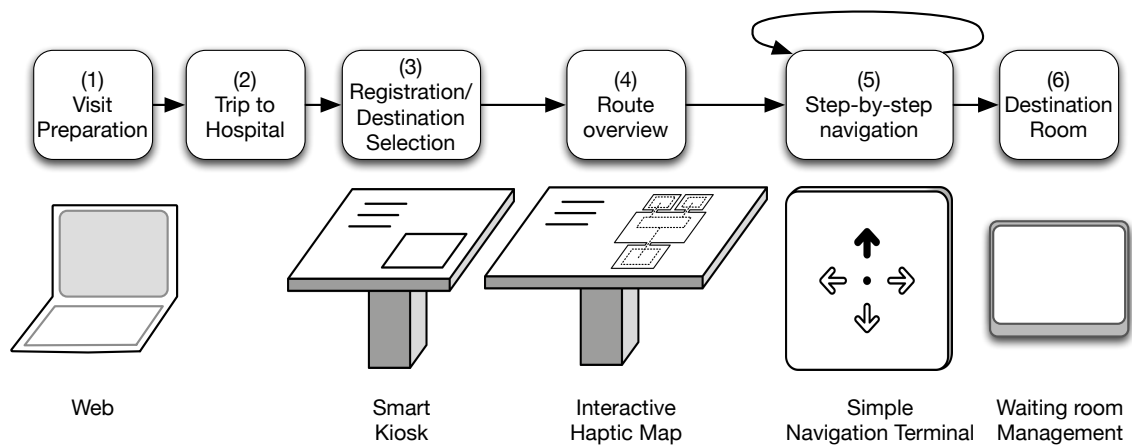


Figure 8.2: Navigation procedure with proposed system.

The course of navigation using our approach is depicted in Figure 8.2. In the first step (1), the user can prepare for his/her visit by using the hospital's web portal. This simplifies the subsequent on-site registration, and enables navigation terminals to adapt their UIs to the user's needs from the very first moment. After reaching the hospital (2), the user enters the hospital building and uses *Smart Kiosk* (3) to register and to plan his/her actual route through the interior of the hospital. After authentication, the UI of *Smart Kiosk* is adapted to the user's needs and preferences. *Smart Kiosk* also takes an image of the user's face to ensure seamless navigation at a later stage. The aim of the *Interactive Tactile Map* (4) is to help users to build their own mental model of the nearby environment, and also to provide an overview of the planned route through this space. Interactive Tactile Maps are therefore placed on central spots in the environment, e.g. on each floor, next to main elevator. The *Simple Navigation Terminals* (5) provide short directional instructions for reaching another navigation terminal or the destination (6). There are frequent *Simple Nav-*

*igation Terminals* in the environment that provide navigation instructions at least at each corridor junction.

In the following text, we describe individual navigation terminals in detail. However, there are some common features shared by all of them. All terminals are equipped with a camera and are connected to a central server. Each terminal also contains a receiver for input from a visually-impaired person's transmitter [scopus1] - a standardized device to be used by the visually-impaired person to trigger various devices in the indoor and outdoor environment. The main purpose is to localize nearby objects or places rapidly, using sonification approach [53]. In our case, the use of this transmitter is optional. However, it can make it simple to locate individual navigation terminals. Each kind of terminal also provides an option to make calls for help if there is any difficulty. The user can also request navigation to other destinations along the route, e.g. to a WC.

As soon as a user approaches any terminal and the corresponding computer vision software (using a camera embedded in the terminal) recognizes his/her face, the user is provided with personalized navigation instructions. A simple arrow shown on the terminal display is sufficient for communicating with most users, but in some cases other interaction methods need to be used. For example, visually-impaired users are provided with detailed navigation instructions on how to proceed to the next navigation point using an audio modality. The change of interaction modality is preferred automatically in the case the User Model is known.

### 8.1.1 Smart kiosk

The main function of the *Smart kiosk* is to register users to the system and link an image of their face with the corresponding user account. *Smart kiosk* also provides

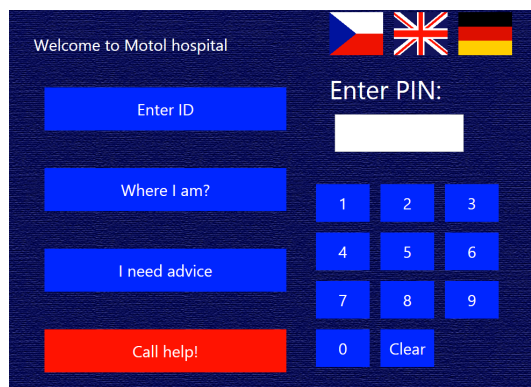


Figure 8.3: Early version of *Smart Kiosk* UI

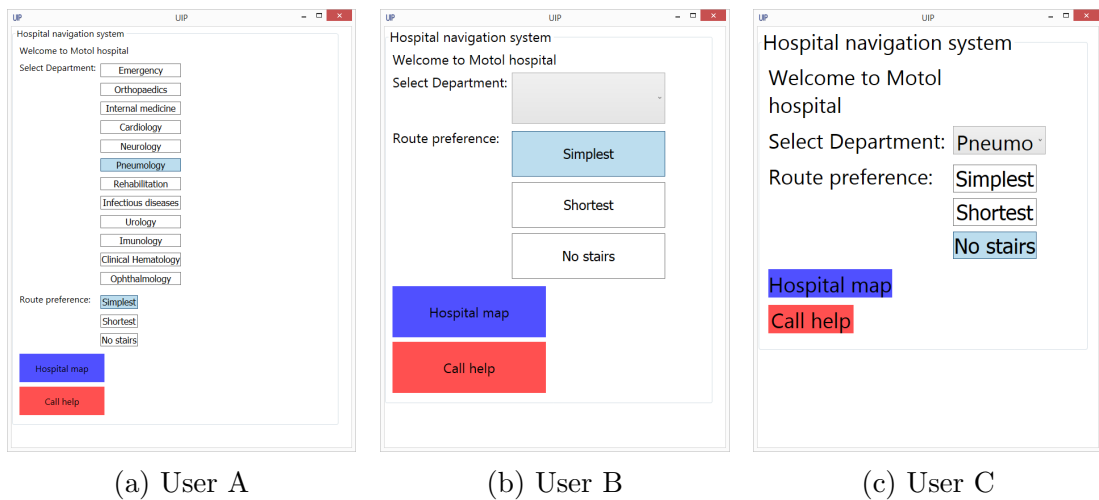


Table 8.1: Context model properties relevant for CUI adaption

Property	DM	UM User A	UM User B	UM User C
font-height	16	1.0	1.5	2.3
target-size	150x50	1.0	1.8	1.0
limitation		–	hand-tremble	low-sighted

a general preview of the whole route to the destination. The UI is adapted to the user’s abilities. Figure 8.3 depicts an early variant of *Smart kiosk* UI. In the default state, the *Smart kiosk* shows simple compromise UI that could be used by the vast majority of users. After user identification, the UI is adapted to the needs and preferences of individual users in case they already provided information necessary for User Model specification.

Figure 8.4 shows an example of context-adaptions performed on the *Smart kiosk*. In this example, we show adaptations for three users with different needs and abilities. Table 8.1 describes context-model properties, that are relevant for CUI adaptations shown in this example. The *User A* does not require any special adaption, note that all values in the UM are in this case 1.0. The *User B* has slightly worsened sight, but also severe issues with the dexterity – hand tremble. This is reflected in the value of *target-size* property, that is in this case 1.8. The *User C* has moderate issues with his sight. This situation is reflected by value 2.3 of the *font-height* property that causes all fonts are 2.3 times larger for the *User C*.

Figure 8.4: Example UI of *Smart kiosk* prototype

### 8.1.2 Interactive tactile map

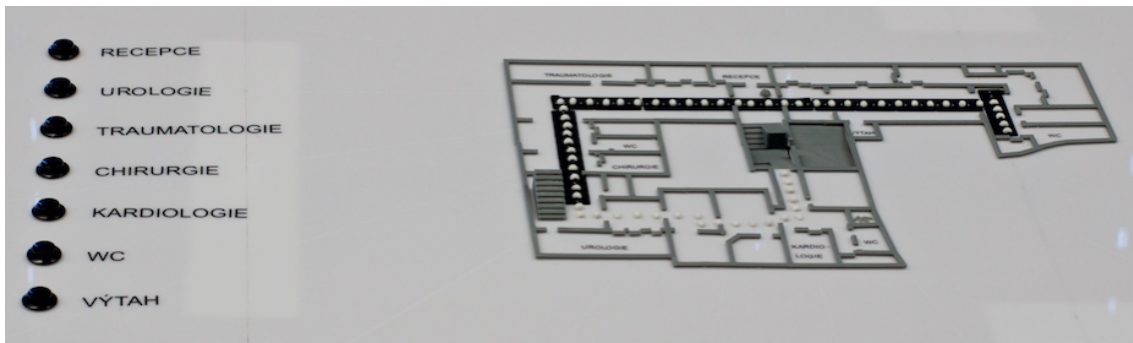
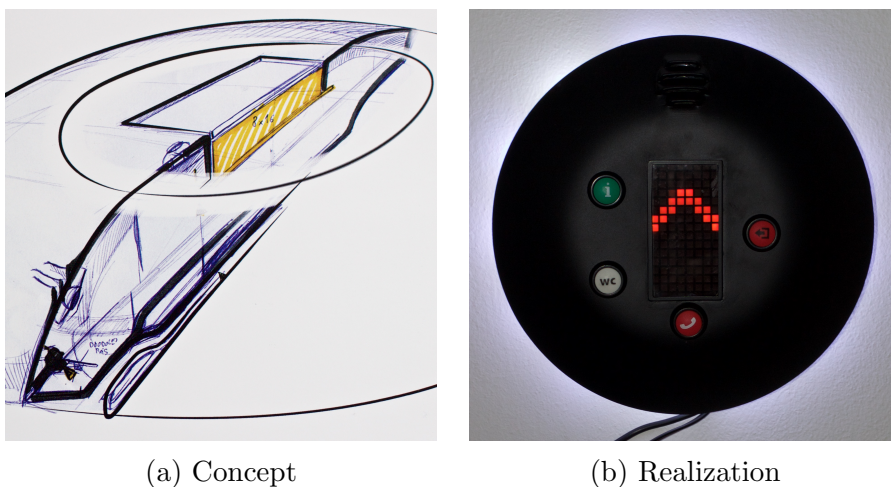


Figure 8.5: Prototype of interactive tactile map

An *Interactive tactile map* provides topological information about large parts of the hospital, e.g. one floor. Prototype is depicted in Figure 8.5. The physical design of the map is tailored for various user groups, in particular for visually-impaired people and seniors. The user can explore the environment depicted by the map using both visual and haptic modalities. The actual route is visually highlighted. Touch sensitive sensors placed along the route are used to detect user interaction – tactile exploration. Using this method, we can provide additional information useful for the navigation. This feature focuses primarily to improve the interaction with visually impaired users.

### 8.1.3 Simple navigation terminal



(a) Concept

(b) Realization

Figure 8.6: Concept and physical realization of final prototype of Simple Navigation Terminal

The *Simple navigation terminal* is the basic building block of our navigation system. The primary aim of the *Simple navigation terminal* is to provide simple directional instructions at the corridor junctions. The interaction with it is carried out as follows. First, the motion of a potential user is detected (1). In the next step, a fast face detection algorithm is performed on the captured image (2). If a face is present in the image, the frame is sent to the server that performs advanced face recognition, resulting in the identification of a particular person (3). After successful user recognition, personalized navigation instructions are provided. However, the visual form is non-intrusive and is adequate for most users. Instructions for visually-impaired users are provided in the form of voice instructions.

This section has described the general design of our navigation system. In the rest of this chapter, we focus on hardware and software implementation of the system. An evaluation follows of the network transmission protocol and face recognition systems that are used. Other aspects of our system, and also user-evaluations of individual development stages of the system, are presented in [scopus1].

## 8.2 Software and Hardware Architecture of Distributed Navigation System

The *UIP* client-server architecture as described in this thesis is used for implementation of the indoor navigation system. *UIP Protocol* is used for both for network communication and for a description of the UI. Figure 8.7 shows adaption of the reference architecture of the *UIP Server* (for more details see section 3.3) for purposes of the navigation system.

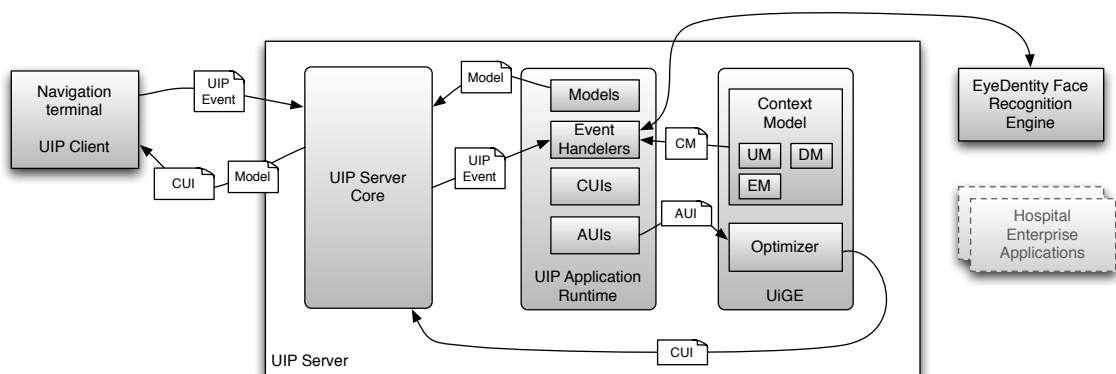


Figure 8.7: Client-server architecture of distributed navigation system

For the purposes of user identification, *EyeDentity Face Recognition Engine* [33] is connected to *UIP Server*. The *EyeDentity* engine finds correspondences between the currently provided image and a set of already known images. The latter group consists of faces to be identified in the incoming video stream. In our case, this group represents individuals that are currently registered as users our navigation system.

*Simple Navigation Terminal* is the basic building block of our navigation system. Its design, see Figure 8.6 is the result of iterative development based on a user-centered design method [2]. Figure 8.8 depicts the internal components of this type of terminal. Four layers can be identified in the terminal. This type of terminal also connects to *UIP Server* (*UIP Server Layer*), but its functionality is limited to a physical interface. It does not support rendering of *UIP Interfaces* - instead it binds the values in data *Models* to its physical components. *UIP Client* itself (the *UIP Client Layer*) runs on Raspberry Pi 2 [108]. This single-board computer provides analog and digital interfaces to connect various peripherals.

The software for the *Simple Navigation Terminal* is implemented using the *.NET framework* [107], in this case running within the *MONO framework* [94]. The reference implementation of *UIP Client* has been extended to enable communication with specific HW components, e.g. *Bi-Color Matrix Display*. A *.NET wrapper for OpenCV – EmguCV* has been used to capture the camera video stream. It also performs motion detection as well as face pre-detection in the video stream. The aim of face pre-detection directly on the terminal is to preserve the capacity of the communication channel and to reduce server utilization.

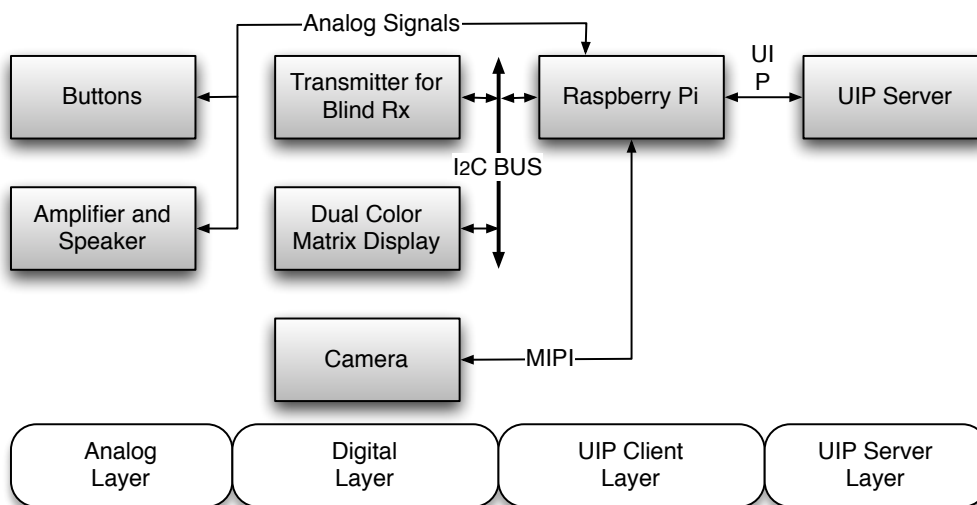


Figure 8.8: Simple Navigation Terminal Components

## 8.3 Conclusion and Contribution

In this chapter, we have presented a prototype of in-hospital navigation system tailored to people with limited navigation and orientation capabilities. This example shows the use of our approach for purposes of complex distributed systems that consists of different clients based on different UI platforms. In this example, there is special focus on adaptive features of our solution and its support for UI accessibility.



# Chapter 9

## Other applications

This chapter focuses on other applications that are only partially based on methods developed in the framework of this thesis. Those selected applications use the methods introduced by this thesis in a novel way that illustrate their flexibility. Figure 9.1 depicts applications described in this chapter. In section 9.1 we show the use of our approach for in a use-case of a form-filling solution based on a current e-governance system. In this case, we show two different methods. Firstly, *UIP CUIs* are directly generated from the proprietary format (Form 602). Secondly, full transformation based on the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline), including the context sensitive UI generation is employed. *GraFooSha* (see section 9.2) is a physical product, that was integrated into the *UIP Platform* as a special *UIP Client*. It does not use automatic CUI generation, but it illustrates the flexibility of our CUI delivery method.

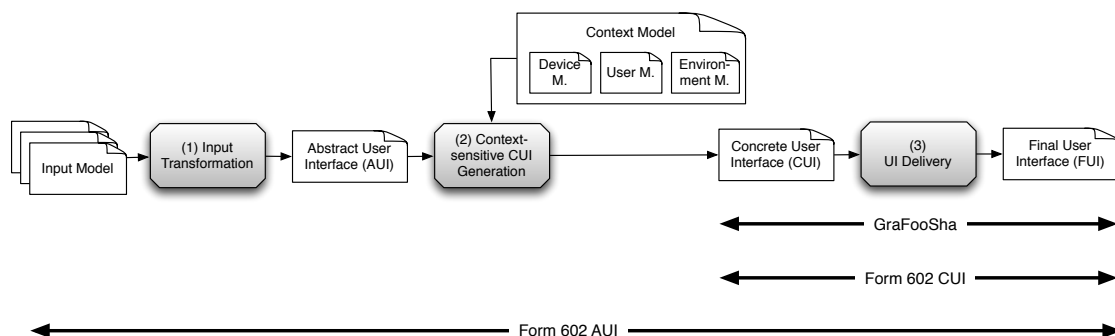


Figure 9.1: Applications described in this chapter in the scope of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

## 9.1 E-governance

In this section, we describe the use of the *UIP Platform* for the purposes of electronic form-filling application. In the framework of this project, the input was derived from the electronic forms description format – *Form602 zfo* [100] (*602zfo*), that widely used in the Czech Republic.

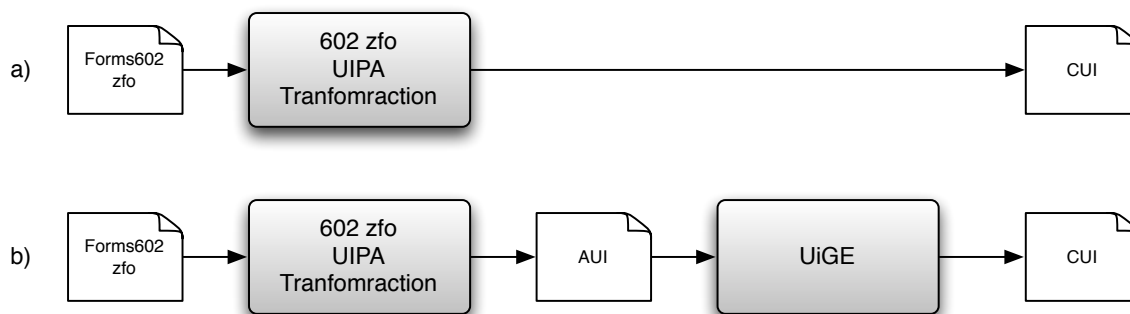


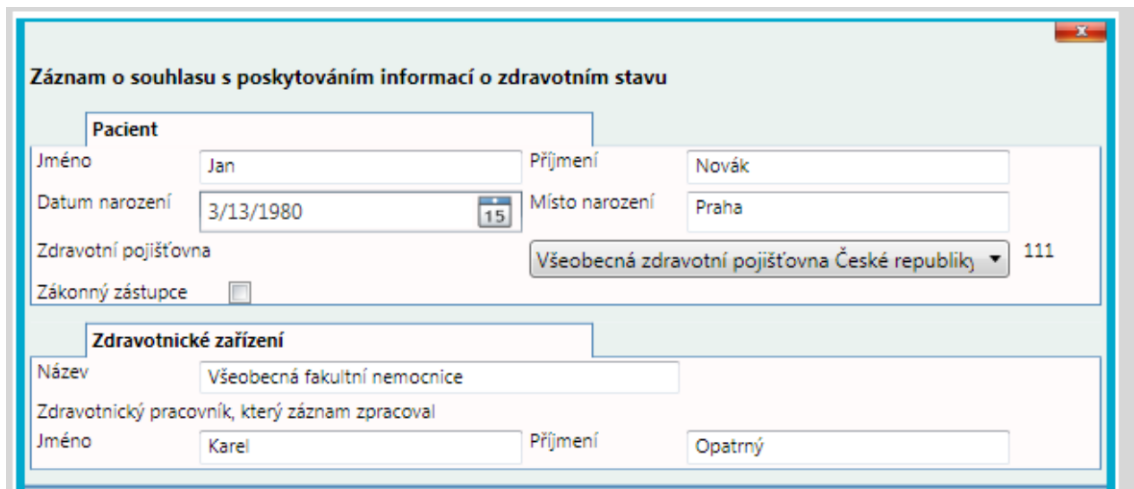
Figure 9.2: Transformation from 602 zfo to *UIP* CUI and AUI

In Figure 9.2 two ways of integration of *602zfo* into the *UIP Platform* are shown. It is possible to transform the *602zfo* format directly into the CUI as depicted in Figure 9.2a. The more sophisticated way is to derive an AUI from the *602zfo* and used context-sensitive transformation into CUI by the *UiGE Pipeline* – Figure 9.2b.

Záznam o souhlasu s poskytováním informací o zdravotním stavu	
<b>Pacient</b>	
Jméno	Jan Novák
Datum narození	13. March 1980
Místo narození	Praha
Zdravotní pojišťovna	Všeobecná zdravotní pojišťovna České republiky
Zákonný zástupce	<input type="checkbox"/>
<b>Zdravotnické zařízení</b>	
Název	Všeobecná fakultní nemocnice
Zdravotnický pracovník, který záznam zpracoval	
Jméno	Karel Opatrný

Figure 9.3: Form rendered using original Form Filler Software (Czech variant of consent form before medical examination)





**Záznam o souhlasu s poskytováním informací o zdravotním stavu**

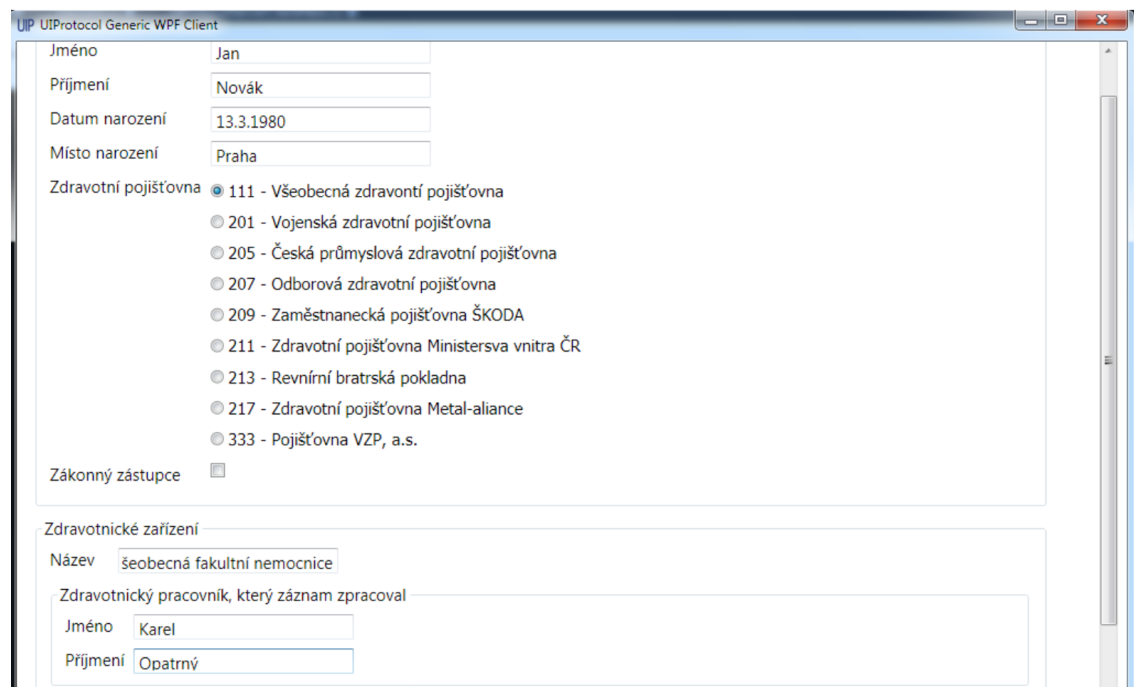
**Pacient**

Jméno: Jan Příjmení: Novák  
Datum narození: 3/13/1980 Místo narození: Praha  
Zdravotní pojišťovna: Všeobecná zdravotní pojišťovna České republiky 111  
Zákonný zástupce:

**Zdravotnické zařízení**

Název: Všeobecná fakultní nemocnice  
Zdravotnický pracovník, který záznam zpracoval  
Jméno: Karel Příjmení: Opatrný

Figure 9.4: Same form rendered using UIP Desktop Client



UIP UIProtocol Generic WPF Client

Jméno: Jan Příjmení: Novák  
Datum narození: 13.3.1980 Místo narození: Praha  
Zdravotní pojišťovna:  111 - Všeobecná zdravotní pojišťovna  
 201 - Vojenská zdravotní pojišťovna  
 205 - Česká průmyslová zdravotní pojišťovna  
 207 - Odborová zdravotní pojišťovna  
 209 - Zaměstnanecká pojišťovna ŠKODA  
 211 - Zdravotní pojišťovna Ministersva vnitra ČR  
 213 - Revnírní bratrská pokladna  
 217 - Zdravotní pojišťovna Metal-aliance  
 333 - Pojišťovna VZP, a.s.  
Zákonný zástupce:

**Zdravotnické zařízení**

Název: všeobecná fakultní nemocnice  
Zdravotnický pracovník, který záznam zpracoval  
Jméno: Karel Příjmení: Opatrný

Figure 9.5: Equivalent form as a result of automatic CUI generation

Figure 9.3 shows an example form that was rendered using the original desktop application. This can be compared with Figure 9.4, that shows the same form, but rendered using the *UIP Platform*, in this case, using desktop client based on .NET framework. Note the capabilities of *UIP Platform* clients to render complex UIs from a single CUI input.

In Figure 9.5, there is depicted a UI that is functionally equivalent to the above described UIs. In this case, the UI was automatically generated from an AUI description. In this case, the visual appearance resembling the original paper forms is not preserved. On the other hand, this solution enables optimization that improves usability and accessibility of the resulting UI.

This section highlighted the capabilities of our approach from the perspective of integration of practically used formats. The added value over the legacy solution is support of various UI platforms in the simple transformation case. The possibility of complex transformation using the context-sensitive CUI generation brings additional advantages for the form filling efficiency even for individuals with disabilities.

## 9.2 GraFooSha: Food Sharing for senior users

*GraFooSha* [*GRAndmaFOOdSHAring*] is a device that provides senior users with access to a food sharing social network. It is a physical device that can be categorized into the domain of Internet of Things (IoT) [39]. Interaction design of this device incorporates deep-rooted concepts the target group is familiar with. *GraFooSha* is an example of an application that does not employ full automatic CUI generation based on the UiGE Pipeline. The current version of the UiGE Pipeline does not support automatic generation of physical UI that can be represented by the *GraFooSha* as an example. At this point, instead, we show a technical realization that illustrates practical capabilities of the *UIP Platform*. In this case, we show that it is capable to efficiently control a device in a role of a physical UI.

With the introduction of modern Information and Communication Technologies (ICT), the social interaction between people moved strongly towards virtual worlds of software-based social networks (like Facebook) which is hardly accessible by seniors. Seniors are typically struggling with social exclusion caused by the loss of friends and relatives and by mobility restrictions determined mainly by their health status. Complex and inaccessible ICT is further deepening their social exclusion.

Food sharing can be an interesting social activity. Especially senior women are used to invite relatives and friends to share food and in such way maintain and

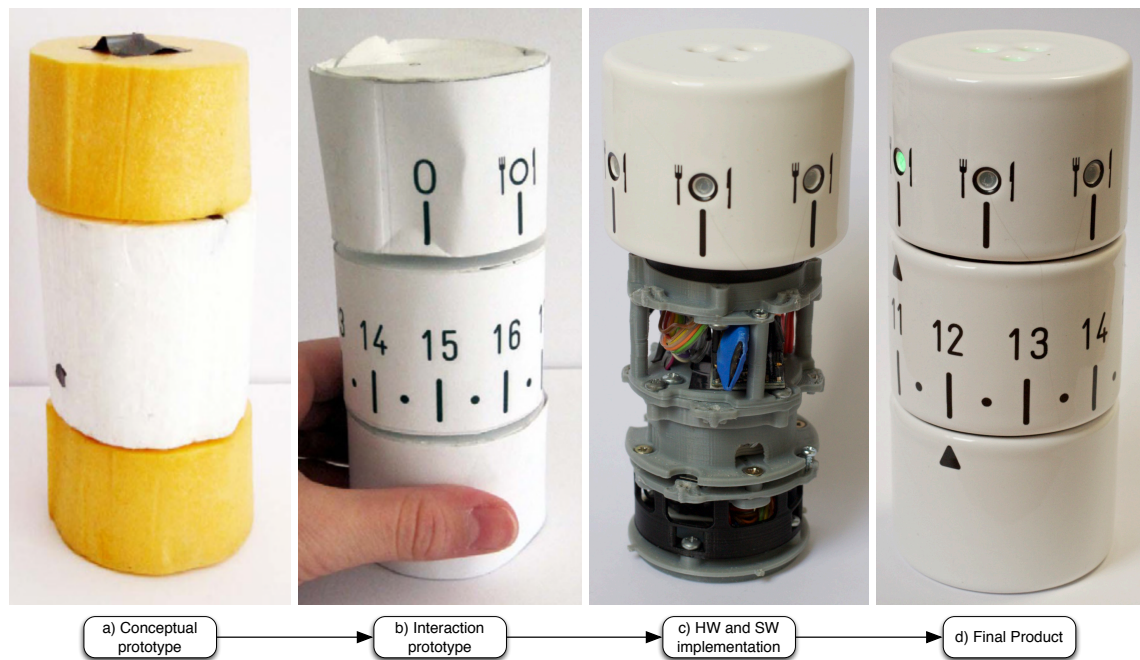


Figure 9.6: Design development of GraFooSha

develop social relations. As there exist several food sharing projects, not rarely supported by ICT, we found as an interesting idea to allow senior women to join such projects without the necessity to learn anything about the ICT and applications used for facilitating the food sharing activities.

Figure 9.6 depicts development of *GraFooSha* design. Basic functional requirements on the device are following: specify of number of meal portions, specify the time the meal will be ready, select meal recipe, and provide meal subscription feedback. Conceptual prototype (a) was used to determine the physical form-factor of the device. Interaction prototype (b) was used for development and usability testing of the corresponding interaction method. Remaining two images – Figure 9.6c-d represents implementation of *GraFooSha* mechanical components and the final product. 3D printing was widely used during the development.

### 9.2.1 Technical realization

Electronic components in *GraFooSha* implement its interactivity and handles communication with the meal-sharing social network. Block scheme of the electronic components interconnection is in Figure 9.7. The *Top Shaft* hosts three RGB LEDs that indicate the selected meal by corresponding color. There are also three RG

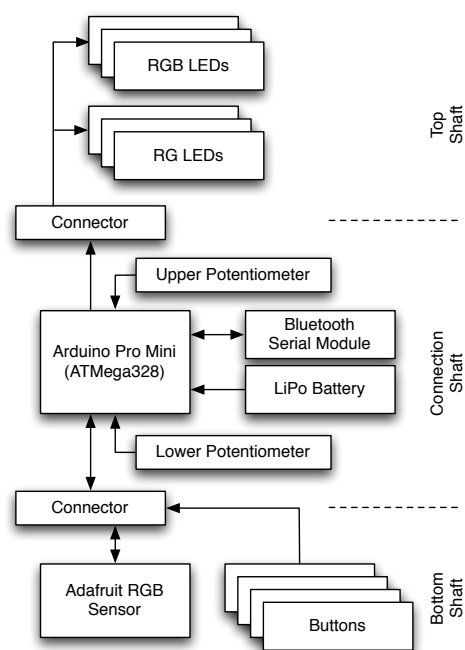


Figure 9.7: Internal Electronic Components of GraFooSha Device

(Red + Green) LEDs to indicate the number of portions offered/reserved. The RG LEDs can be controlled individually. The top part is connected to the central *Connection Shaft* using a 10-pin connector. Rotation angle of individual shafts is measured using two potentiometers (lower and upper). All information is handled by *Arduino Pro Mini* micro-controller.

Using a *Bluetooth Module*, *GraFooSha* connects to the *UIP client gateway* via an emulated serial port. A *Lithium Polymer* Battery hosted in the *Connection Shaft* powers the device. The *Bottom Shaft* hosts an RGB Sensor and four buttons. The RGB sensor is used to determine the color that corresponds to a particular meal recipe in the cookbook. Each meal is represented by corresponding color-code. Color scanning is activated when press is detected using any of four buttons that detect click-like motion between *Bottom and Central Shaft* of the device.

In case of the final product, *GraFooSha* will be connected to the food-sharing network via a Bluetooth communication link with its base-station. This base station will be connected to the Internet and act also as an inductive charger. Currently, the base station is simulated by a computer. UI of a simulated food-sharing network is implemented using our *UIP Platform*.

Figure 9.8 shows the integration of the *GraFooSha* and the *UIP Platform*. *GraFooSha* uses a serial link communication over the bluetooth link to commu-

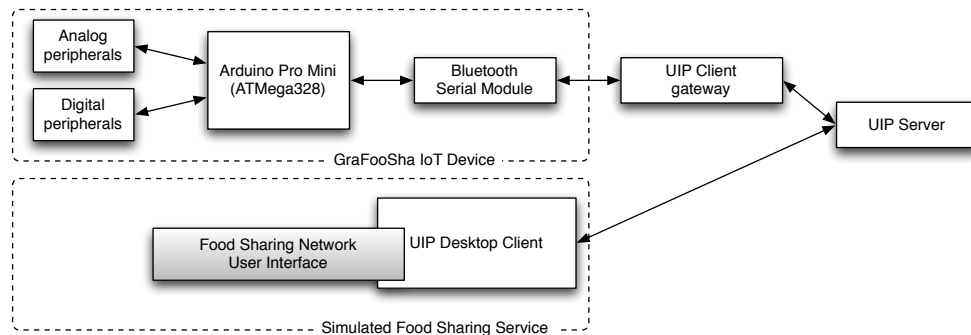


Figure 9.8: Integration of GraFooSha IoT device into the *UIP Platform*

nicate with a gateway application. From the perspective of the *UIP Platform*, this gateway application represents a *UIP Client*. Interaction with the physical device, then results in generation of *UIP Events* that can be handled on the *UIP Server* in the standard manner. In the opposite direction, *UIP Model updates* affect the states of the physical device. For example, RGB LEDs on *GraFooSha* can show any color code sent by the *UIP Server* in the form of a *UIP Model update*.

Another *UIP Client* has been used to render UI of the simulated food sharing network as shown in the bottom part of Figure 9.8. This UI corresponded to interaction from the food consumer perspective. In case of this example, the *UIP Platform* has been used for rapid development of complex solution consisting of different kinds of networked devices. For more details about *GraFooSha* project refer to [other2].

### 9.3 Conclusion and Contribution

In this chapter, we focused on additional interesting applications of approach introduced in this dissertation thesis. Section 9.1 described a form-filling solution in the domain of e-governance. UIs rendered using our multi-platform approach are compared to UIs of a legacy single platform form-filling application. Section 9.2 shows an application of our approach in the domain of Internet of Things (IoT) [39]. However, this application does not employ the full automatic CUI generation. As a subject of the future work, it would be also possible to develop a transformation method that will use pre-prepared or 3D-printed components to automatically generate physical interactive devices that play a role of a User Interface. Using these examples, we want to highlight communication capabilities and versatility of our approach. Our approach was used for purposes of more applications, e.g. [scopus3], in this chapter, we highlighted the most important ones.



# Chapter 10

## Evaluation

This chapter focuses on evaluation of the methods introduced in the framework of this dissertation thesis. Firstly, we focus on evaluation of generated UIs from the perspective of target users. section 10.1 describes the evaluation of the Context Model. In the next section 10.2, perceived quality of resulting UIs is evaluated. Follows an evaluation from the perspective of application developers. On a case of application that uses data-persistence, we compare the development efficiency of our approach in comparison with a manual development method. An evaluation of network transfer capabilities of *UIP* is also part of this chapter. It is described in section 10.4. This chapter also focuses on performance of complex actions performed directly on *UIP Clients* with limited computational power.

### 10.1 Evaluation of context model adaptations

The goal of this test was to evaluate the design validity of the proposed Context Model (CM) and to gain the user feedback for its eventual improvements. The research question was whether it is possible to use separated Context Model (CM) sub-models (DM, UM, ATM) as described in chapter 5.

The study was conducted with twelve participants (9 male, 3 female, age 23 - 59, mean = 35 years). The study procedure was as follows. Four different CM properties were the subject of the study:

- *Font-size* - Size of the font used for common user interface elements like labels, buttons etc.
- *Element-spacing* - Minimal distance between particular user interface elements.

- *Target-size* - Minimal size of interactive areas to be reached by mouse pointing or tapping.
- *Line-width* - Minimal width of lines. This value influences both UI components (e.g. group box, separator) and vector graphics figures.

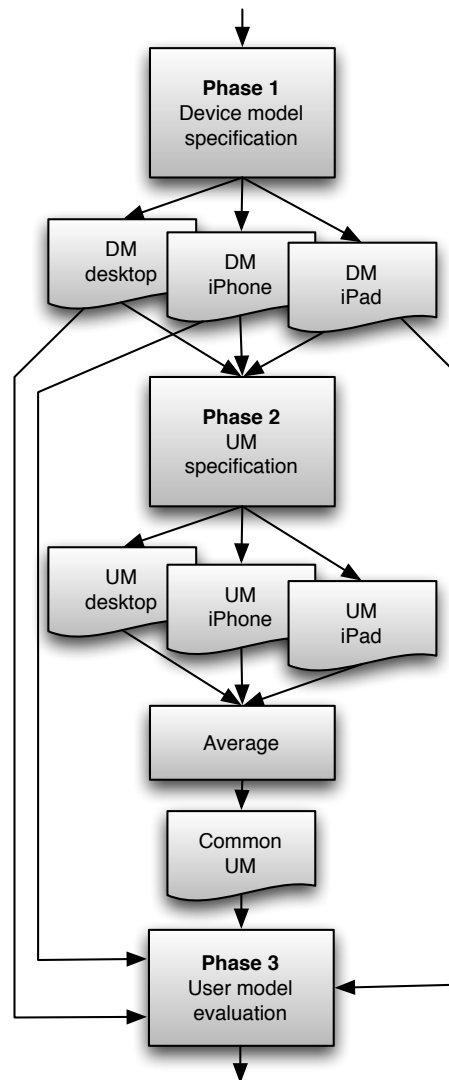


Figure 10.1: Plan of the three-phase Context Model (CM) evaluation study.

Three different target devices were used for the study: standard desktop computer, tablet computer (Apple iPad) and smartphone (Apple iPhone). The study consisted of three phases as depicted in Figure 10.1:

1. *Phase 1 - Preparation - Device Model (DM) definition*: In this phase the absolute values in the DM were refined on a sample of 8 users. A particular



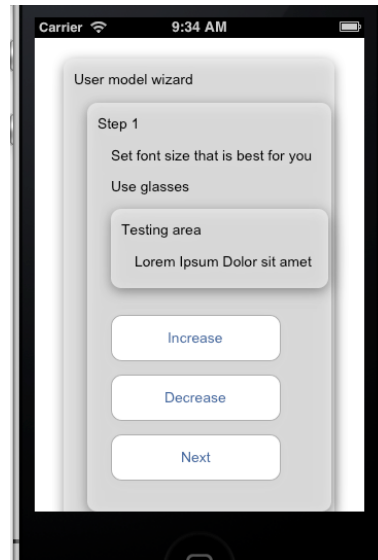


Figure 10.2: Example of user interface used in the study

value in DM has been computed as average of values determined by individual users as best. In this phase the users have no substantial vision or dexterity impairment and use glasses if necessary.

2. *Phase 2 - User Model (UM) and Assistive Technologies Model (ATM) measurement*: In this phase all three test devices were presented to 12 users. The order of tests with particular devices was scrambled to exclude possible bias (each group of 6 users has all possible permutations). For each device a particular user was presented with individual tested context properties. At the beginning, each property has default value for particular devices defined in the DM. Individual users were asked to set a value of each context property that preferable fits their preferences and needs. Users that wore glasses were firstly asked not to use them. Users set optimal values using simple increase/decrease buttons (see Figure 10.2). In this way the preliminary UM was derived for each device. Finally, the overall UM was computed as average of preliminary UMs derived for particular devices. The ATMs of glasses for particular users were derived in the same manner. Individual properties in the ATM were computed as the ratio between value measured using the glasses and original value in particular UM (without glasses). For properties like font-size or line-width is the corresponding value in glasses ATM typically less than 1 - total context property value can be smaller when using glasses.

3. *Phase 3 - User Model (UM) and Assistive Technologies Model evaluation:* The subject of the final phase was to indicate the validity of the Context Model. This phase focused on degree of user satisfaction and error of the overall UM. Users were presented with the same device set, but with UM derived for each individual user in the previous phase. At each step the users were asked to assess the current value using the criteria shown in Table 10.2. In this phase two users from each group (see hereinafter) served as control sub-group and were presented with the original default UM (all properties has value 1.0 - no effect to default properties in DM).

There were three user groups - users that does not need glasses (Group 1 - UM evaluation), users that need glasses but currently didn't use them (Group 2 - UM evaluation) and users that need glasses and used them (Group 3 - UM and ATM evaluation). The latter two groups consisted of the same people. The UM and ATM were evaluated against original UMs and ATMs measured for particular devices before the average overall models were constructed. Figure 10.1 shows the results. Generally the tested UI properties were well accepted by the user audience (no rating was worse than 3, average rating was 1.35). The most significant error was recorded for the font-size property (Group 2, average error 23.67%, maximum error 57%). This property was assessed as the least-suitable by the users as well (average rating 1.69, worst rating 3).

The user testing indicated the validity of our Context Model (CM) concept, however a further study on a larger user audience is necessary for a statistical evaluation. Follow possible improvements to the context model that emerged from this study. The font-size and line-width properties should not be computed as average, but the biggest coefficient measured should be used. The target-size property should be in

Table 10.1: Results of CM evaluation study

Group	Tested model	Need/Used Glasses	Type	Evaluation metric	Value	Font Size	Spacing	Target size	Line Width
Group 1	UM	NO/NO	Test group	Preliminary UM deviation	Average	2,22%	8,22%	5,22%	9,67%
				Maximum	13,00%	20,00%	20,00%	27,00%	
			Control group	Rating	Average	1,33	1,33	1,25	1,42
				Maximum	2	2	2	3	
Group 2	UM	YES/NO	Test group	Preliminary UM deviation	Average	23,67%	14,33%	16,33%	10,44%
				Maximum	57,00%	33,00%	40,00%	33,00%	
			Control group	Rating	Average	1,69	1,23	1,69	1,23
				Maximum	3	3	3	2	
Group 3	UM, ATM	YES/YES	Test group	Preliminary UM deviation	Average	6,33%	9,87%	17,83%	13,06%
				Maximum	30,00%	20,00%	60,00%	33,00%	
			Control group	Rating	Average	1,08	1,50	1,25	1,17
				Maximum	2	3	2	2	
Group 3	UM, ATM	YES/YES	Control group	Rating	Average	1,17	1,50	1,00	1,50
				Maximum	2	2	1	3	

Table 10.2: Subjective assessment of context model properties

Rating	User interface property value is
1	exactly as desired
2	almost as desired
3	usable but it should be rather changed
4	barely usable, it should be substantially changed
5	not usable, it must be dramatically changed

UM and ATM separated into more properties, in this case there should be a specific property for mouse pointing devices and devices where objects are selected by finger tapping.

## 10.2 Evaluation of perceived quality of generated UIs

The aim of this user study was to compare UIs generated by using the Context-sensitive Automatic Concrete User Interface Generation Pipeline with manually implemented simple web-based UIs.

The testing procedure was as follows. Twelve participants (10 male, 2 female, aged 22 - 48, mean = 30 years) were hired to evaluate the quality of UIs generated using our method. Three different UI platforms were used during the study – desktop PC, tablet PC and smartphone. At each platform, the participants of the study were presented with UIs generated using our Context-sensitive Automatic Concrete User Interface Generation Pipeline and a web UI with a similar functionality. The UIs were assessed subjectively in terms of *comfort*, *efficiency* and *aesthetic quality*. The participants of the study were skilled ICT users who use a computer as their primary work tool. With one exception, all users have a university degree. Each participant was asked to fill in an ACM-ICPC registration form, using both a web browser and UIP client on each platform. The order of the tests was scrambled to avoid the possible bias.

Afterwards, the participants were asked to evaluate all six presented UIs for comfort, efficiency and aesthetic quality, on the Likert scale [74] (1 to 5, where 1 is the best score). The Wilcoxon rank-sum test [56] was used to compare the effect of each UI on *comfort*, *efficiency* and *aesthetic quality*. The Wilcoxon rank-sum test is

Table 10.3: Subjective evaluation of form UIs.  $M(x)$  is median of Likert scale assessments.

Device	UI aspect	M(web)	M(UIP)	Statistical result	Sig. difference
Desktop	comfort	2	1	$W = 108, p < 0.05$	yes
	efficiency	2	1	$W = 100, p > 0.05$	no
	aesthetic quality	4	2	$W = 115.5, p < 0.05$	yes
iPhone	comfort	4	2	$W = 121, p < 0.005$	yes
	efficiency	4	2	$W = 114, p < 0.05$	yes
	aesthetic quality	4	2	$W = 134.5, p < 0.001$	yes
iPad	comfort	3	2	$W = 109, p < 0.05$	yes
	efficiency	3	2	$W = 110, p < 0.05$	yes
	aesthetic quality	4	2	$W = 126, p < 0.005$	yes

a statistical test that compares two samples and assesses whether their population mean ranks differ. It is used as an alternative to the paired Student t-test in cases when the population cannot be assumed to be normally distributed [40]. The results of the statistical evaluation are presented in Table 10.3.

The user study showed that the generated UIs provide a better subjective user experience for comfort, efficiency and aesthetic quality. On the desktop PC, the usage comfort was evaluated as better for the UIP client ( $M = 1$ ), though both UIs provide good results. There is a non-significant difference in efficiency between UIP client ( $M = 1$ ) and web UI ( $M = 2$ ), so their performance can be evaluated as comparably well. The aesthetic quality was evaluated as significantly better for the UIP client ( $M = 2$ ) than for the desktop web UI ( $M = 4$ ).

Using the iPhone smart phone, all three metrics were evaluated as significantly better for the UIP client. For all three metrics, the medians of the Likert scale values were 2 for the UIP client and 4 for the respective web UI. For the iPad tablet, all three metrics were also evaluated as significantly better for the UIP client. For the web UI, the medians for comfort and efficiency metrics were 3, and for aesthetic quality the median was 4. The UIP client provided better results with median = 2 for all evaluated metrics.

Generally, the *UiGE Pipeline* provided UIs that were highly positively rated in the user study. The most significant advantage over the web UI was on the iPhone

platform. In this case, the UI generated by the *UiGE Pipeline* was based on an easy-to-navigate iOS Table View. By contrast, the web page required a lot of scrolling and zooming, which worsened its usability.

### 10.3 Development and Maintenance Efforts

This section describes evaluation of our solution based on application source code audit of data-driven applications as described in chapter 7. The primary focus is on development effects and amount of information that needs to be restated in either case.

The study is further evaluated from the perspective of development and maintenance. We consider a manually developed system and compare it with a system built using our approach. The persistence model of the considered application consists of 7 entities. For both applications considered here, the model consists of 370 physical lines of code (LOC). With the manual approach, it is necessary to implement XML forms for the UIP, and it consists of JavaScript (JS) references and a JS library. The XML forms have 703 LOC, while the JS references and library have 300 and 446 LOC.

With our approach, the persistence model is extended with additional marks, resulting in 96 additional LOC for the persistence model. Generic UIP configuration, templates and Event handlers are designed for AspectFaces library, and all these are applicable to different UIP projects. The UI part is generated through persistence model inspection and transformation. To deal with text labels, we apply them in

Table 10.4: Comparison of manual and code-inspected approach regards the size of code.

Component	Manual approach (LOC)	AF approach (LOC)
Persistence model	370	370
Extension to the persistence model	-	96
Text label properties	-	108
UIP XML forms	703	-
JavaScript library	300	300
JavaScript in view	446	-

the text property file rather than in forms (108 LOC). Table 10.4 summarizes the efforts. In order to develop the project manually, we need to implement 370 LOC of the persistence model, 703 LOC of XML with weak type safety, and 746 LOC of JS. With our approach, we only need to implement 466 LOC of the persistence model, define text properties (108 LOC) and use 300 LOC of the JS library.

It must be considered that when we develop the UIP project manually, the entire presentation source code in addition to Event Handlers must be developed. Especially in the XML part, with forms, there is a burden of restated information in a weak type safety, so future changes to the persistence model must also be manually applied in the XML part. By contrast, with our approach the changes take place only in a single location, the persistence model. We must consider that there is a reduction not only in the source code, but also in the coupling among different subsystems. In addition, the time dedicated to both development and maintenance is reduced, because the UI part adjusts to the information already captured in the persistence part, and no manual restating takes place.

In this section, we have shown evaluation of fusion solution of *UIP Platform* and application source code audit based on *Aspect Faces*. A synergy effect of this fusion resulting in further reduction of the amount of necessary source code and information coupling is clear from the results.

## 10.4 Evaluation of network transfer protocol

This section presents an evaluation of *UIP Protocol* from the perspective of data-transfer over a computer network. The focus is on transfer of bitmap images encoded into *UIP Events*. The corresponding use-case was user identification using face-recognition for purposes of hospital indoor navigation system (see chapter 8). Therefore, another evaluated aspect was the performance of face detection (existence of any face in the image) directly on *UIP Client* running on *Raspberry PI 2* single board computer.

In our experiment, the dependent variables to be observed will be frame size, frame transfer time and the error rate of the face recognition system. The independent variables, which will be altered, are the resolution of the input stream (80x60 to 640x480 pixels, or up to FullHD for a real camera stream) and the existence of pre-detection of faces in the input stream.

Table 10.5 shows the relationship between frame-transfer time and image resolution, and also the influence of face pre-detection. The data in Table 10.5 are average

Table 10.5: Frame transfer time (Simulated stream)

Resolution	Size [KB]	Frame transfer time [ms]	
		Face pre-detection Enabled [ms]	Disabled [ms]
640×480	32.70	984.67	98.25
320×240	11.91	444.07	32.85
160×120	4.72	339.29	16.42
80×60	2.06	301.28	10.14

Table 10.6: Frame transfer time (Real-time video)

Resolution	Time [ms]	Size [KB]
1920×1050	850.17	111.76
800×600	221.11	39.30
640×480	151.17	30.63
320×240	56.06	10.74
160×120	31.64	4.25
80×60	31.61	1.91

values from transferring a sequence of 150 frames. The images were stored in *UIP Client* flash storage in the form of a sequence of 150 png images. This sequence was pre-loaded into the operating memory before the transfer began. The current communication channel transfers each frame separately embedded into a *UIP Event*. As *UIP* is based on *XML*, the image data must be encoded into a valid form. In our case, the valid form is Base64 string. The data show that the current communication channel is sufficient for transferring the video stream up to resolution 640x480 with more than 10 FPS (Frames Per Second). The overhead of Face pre-detection is significant - it prolongs the frame transfer time 10-fold for resolution 640x480, and for lower resolution the impact is even greater.

Table 10.6 depicts the frame transfer times and the frame sizes using real-time video (captured by a camera attached to the *UIP Client* device). There is another overhead, in comparison with the simulated stream, because the images have to be processed by the attached camera. First, the images were re-sized from higher resolution before the frame was transmitted. It is clear from the data that the current communication channel is not adequate for transferring the FullHD (1920x1050) stream. In this case, it can achieve only about 1 FPS.

The first outcome of our experiment is that face pre-detection on relatively slow devices like *Raspberry Pi* can cause delays that will have a significant impact on the interactivity of the system. This is acceptable if there is a need to reduce significantly

the amount of data transferred over a network connection. In our case, we assume that there is a well-dimensioned network that will be able to transfer streams from all connected clients.

The second outcome of the evaluation is that the network channel used here can seamlessly transfer image streams at the frame rate of about 10 FPS for resolution up to 640x480. To transfer video streams with higher resolution, it would be necessary to set up a special network stream rather than using the current *UIP Event* infrastructure. A *User Datagram Protocol (UDP)* protocol stream can be constructed using the current *UIP* infrastructure.

## 10.5 Conclusion

In this chapter, there was described evaluation of our approach from different perspectives. section 10.1 focused on evaluation of the Context Model (CM) adaptations from the user perspective. The outcome of the CM evaluation indicated the validity of the CM concept as described in chapter 5. However, the evaluation also stated demand for additional possible improvements of the Context Model.

Section 10.2 describes evaluation of automatically generated UIs on an example of registration form for international programming contest. Generated UIs have been compared with standard web application with similar functionality. Generally, our *UiGE Pipeline* provided UIs that were in the user study mostly rated better than web-based UIs.

In section 10.3 we focused on evaluation of our approach from the perspective of application developers. Our solution based on application-source code audit and aspect-based AUI derivation proved to be efficient in comparison to a manual approach.

The test described in section 10.4 evaluates *UIP Protocol* from the perspective of data transfer. The outcome is that current *UIP Event* infrastructure is robust enough even for transferring real-time videos up to a resolution of 640x480 pixels. This part of the evaluation also showed lower performance of complex actions carried out on *UIP Clients* with limited computational power.



# Chapter 11

## Conclusions and Future Work

This chapter summarizes the outcomes of this dissertation thesis and proposes possible future work. The introduction contains the basics about automatic UI generation, as well as the statement of motivation for this work. The subsequent analysis resulted in the statement of objectives. A list of those objectives as stated in the introduction, see section 1.2, follows. Here, we discuss the achievements of this dissertation thesis in relation to those objectives.

1. Definition of a methodology for an *Automatic Context-sensitive Generation of Concrete User Interfaces*.

The primary objective of this thesis was the development of a methodology for the *Automatic Context-sensitive Generation of CUIs*. Chapter 2 summarized related approaches and methods with a similar aim; many of which bring interesting concepts suitable for automatic CUI generation. However, most of them also have substantial shortcomings emerging from the limited UI adaptations they offer on one hand and the complicated development and maintenance caused by complex coupled input models on the other.

This primary objective has been achieved by the development of the methodology described in chapters 3 - 6. Our methodology focuses on the realization of individual stages of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline). Chapter 4 describes the immediate input to the UIP User Interface Generator (UiGE) – the AUI structure and methods of its derivation from other input models. Chapter 5 describes Context Model (CM) that attributes the automatic CUI generation.

In chapter 6 we focus on the automatic CUI generation itself, and on the corresponding CUI optimization method. Our CUI generation approach requires a method to automatically assess the quality of generated UIs in accordance with optimal metrics. A survey of potentially suitable methods for automated UI evaluation is described in section 2.5. The result of this survey is that most methods still require the presence of human experts or need to track user activity while interacting with a real (final) application. The automated heuristic evaluation mostly focuses on a specific application domain (e.g. Web). The process of CUI generation and optimization is described in chapter 6. For the purposes of our approach, we have developed an optimization metric that reflects the number of steps the user needs to carry out to perform typical operations with the resulting UIs. For more complex optimization cases we have defined *optimization heuristics*. Using this method the optimization process can be configured for specific application domains while preserving its generality.

Chapter 3 describes the theoretical foundations of the corresponding *UIP Platform*, our User Interface Description Language (UIDL) and the CUI delivery method.

2. Modeling and Implementation of this methodology in a form of the Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline).

The methodology described in chapters 3 - 6 has been implemented as the *UIP Platform*. The use of this platform for practical application development is described in chapters 7 - 9. Furthermore, the *UIP Portal* described in appendix A focuses on supporting developers that use the *UIP Platform* as well as to serve as a runtime engine for *UIP Applications*.

3. Integration or Development of a *Context Model (CM)*. An existing suitable *Context Modeling* method could be adapted. Alternatively a novel context-modeling method that suits our requirements regarding the development efficiency and consistency with other UiGE Pipeline components can be developed.

An important attribute of this dissertation thesis and our CUI gen-

eration method is the context-sensitivity. Section 2.4 contains a survey of related context modeling methods. Various approaches are suited for specific application domains and these approaches also often require complex ontological deceptions. Such complexity can cause development and maintenance difficulties. In chapter 5, we introduced our context modeling method. It is suited to be easily integrated with other components of the Context-sensitive Automatic Concrete User Interface Generation Pipeline. Individual sub-models like User Model, Device Model, Environment Model or Assistive Technologies Model are maximally mutually independent. Structure of the Context Model is also less complex than the structure of most context-models based on ontologies. Our Context Model can also describe specific abilities of the user in the same way as their limitations.

4. Integration or Development of a *UI Description and Delivery Language/Method*. It should be an integral part of the UiGE Pipeline implementation and support high development efficiency.

A prerequisite for our approach was a sufficient method for a UI description on more levels of abstraction. Our survey on UI description languages has shown that several universal User Interface Description Languages do not offer a satisfactory level of abstraction. On the contrary, some languages allow UI definition solely on the abstract level. Chapter 3 describes our method for UI description and delivery. It enables delivery of UIs to various types of client devices that use different UI platforms. A major contribution of this method is that it simultaneously aggregates all application logic on the server-side, whilst it supports the use of native UI elements on individually supported platforms. Additionally, our method brings further advantages in terms of extensibility and modularity. User Interface Description Language introduced as part of this method supports client-server communication and UI description on an abstract level, as well as on a concrete level.

In section 4.1, we introduce our Abstract User Interface notation and focus on several methods of its derivation from other input models.

5. Minimization of the amount of work required for UI development and maintenance in case of complex multi-platform, context-sensitive UIs.

This objective has been mainly addressed by the principles on which the Context-sensitive Automatic Concrete User Interface Generation Pipeline design is based. Our automatic CUI generation method minimizes the amount of work required from application developers. See analysis of development efforts in section 10.3. Furthermore, we supported the developers by addressing the following sub-objectives.

- (a) Provide developers with supporting tools and usage guidelines that will help them to deal with the UiGE Pipeline.

The UiGE Pipeline has been implemented in a form of the *UIP Platform*. An important step for enabling general developers to use the *UIP Platform* was the definition of the *UIP Application* format as described in section 3.3.2. The following tools have been introduced to support *UIP Application* developers.

In section A.2, we introduced the *UIP Visual Editor* that supports development of *UIP Applications* on the basis of a visual development environment. *UIP Visual Editor* supports both AUI and CUI design. Further details about the *UIP Visual Editor* are in appendix A.2.

Another direction to support developers of *UIP Application* was providing a simple solution to manage *UIP Applications*. *UIP Portal* is a web application to support the *UIP Platform* development. It enables management of *UIP Applications*, management of the corresponding *UIP Server* instances that serve as a runtime for *UIP Applications*. Furthermore, the *UIP Portal* contains various resources related to the *UIP Platform* development. Details about the *UIP Portal* are in appendix A, section A.1.

- (b) Development of an *input transformation* method for input derivation from existing models used in practice.

This objective has been addressed by theoretical methods described in chapter 4. Motivation for the development of methods

to derive AUIs was to further simplify the UI development and reduce the amount of information-restatement. Furthermore, the adaption of existing established methods used in the industry might simplify adaption of our approach.

Section 2.2 focuses on related methods to efficiently derive AUIs from other input models. According to our study, an aspect-based approach can be used for effective input derivation for data-oriented applications. Namely, we focus on application source code audit of data-driven applications using aspect-based approach, on the theoretical level, this transformation is described in section 4.3. Chapter 7 describes the practical application of this method. Another direction was to focus on two different domain-specific input models in the domain of an intelligent household. In section 4.4.1 we focused on the URC platform, and in section 4.4.2 on the OpenHAB platform.

## 6. Evaluation of the proposed solution.

The theoretical methods behind the *UiGE Pipeline* and its implementation – the *UIP Platform* have been evaluated from different perspectives. The evaluation of the essential concepts and methods is described in chapter 10. The following sub-objectives focus on different perspectives of our approach evaluation.

### (a) Demonstration of its usefulness for purposes of practical applications.

Several examples described in this dissertation thesis point to the validity of our approach and highlight its practical usability and utility. In chapter 7, we presented an example of how the aspect-based application source-code audit can be used to generate context-aware UIs at runtime. The proposed method significantly reduces the amount of the replicated source-code, which simplifies both development and maintenance. This example illustrates the use of our approach in the broad extend form complex input derivation through context-sensitive optimizations to the UI delivery to different UI platforms.

Chapter 8 shows the use of the UiGE Pipeline for purposes of complex adaptive indoor navigation. More precisely, this ex-

ample represents an in-hospital navigation system tailored for people with limited navigation and orientation capabilities. Different types of adaptive navigation terminals based on various technologies represent *UIP Clients* in terms of our approach. This example leverages the adaptive features of our approach and its versatility.

Finally, chapter 9 lists applications that partially used our approach. In section 9.1, we highlighted the use of our approach for a form-filling solution the domain of e-governance. The UIs rendered using our multi-platform approach are compared to UIs of a legacy form-filling application. Section 9.2 shows the application of our approach in the domain of the Internet of Things (IoT) [39]. More precisely, this part focuses on the integration of a physical product that helps senior users to interact with a food-sharing service. Although this example does not use the full capabilities of automatic CUI generation, it shows the communication capabilities and versatility of our approach.

- (b) Evaluation of generated UIs from the perspective of relevant target users.

This objective has been addressed by the evaluation of the perceived quality of the automatically generated UIs in section 10.2. This test showed the advantages of our solution over Web with similar functionality. The practical feasibility of the proposed Context Model adaptations have been evaluated in section 10.1. This evaluation pointed to a practical usability of our approach for generation of context-sensitive CUIs. Practical applications of our approach have also been evaluated with their respective target user audience.

- (c) Evaluation of related development support tools from the perspective of their users (the developer perspective).

This objective has been partially addressed by an evaluation of the development and maintenance efforts described in section 10.3. This test showed that our approach enables a significant reduction of the amount of source code that needs to be implemented in comparison with the manual approach. Our approach also enables a reduction of the amount of code that needs to

be restated. This could also reduce the amount of errors during application development and maintenance.

From the usability testing perspective, only the *UIP Visual Editor* (see A.2 in appendix A) has been already evaluated. Details about its evaluation and development based on the User Centered Design [2] are in section A.2.1.

This section summarized the results of this dissertation thesis and the extent to which its particular objectives have been addressed. Most of those objectives have been addressed fully; however, there is also room for further development and evaluation. The following section focuses in detail on proposed future development and a statement of possible subsequent research direction.

## 11.1 Future Work

The UI generation approach based on UiGE Pipeline introduced in this thesis presents an effective solution to various modern ICT problems on one hand. On the other hand, it also implies new research directions, as well as challenges for technological improvements. This section highlights those that are most important.

Additional evaluation of some methods developed in the framework of our approach should be conducted. Our concept of UI optimization heuristics should be evaluated to a broader extent. There is also a need for further qualitative evaluation that compares UIs resulting from our UiGE Pipeline with UIs generated by a comparable state-of-the-art method.

From a *UIP Application* developer perspective, further usability evaluation of supporting tools should be conducted. There is also a need for a statement of development guidelines and best practices. An effective dissemination strategy of our approach should be also taken into account.

From a technological point of view, our approach should be integrated with new promising methods and technologies that have been developed in parallel to our approach; from our current perspective, the most important is a transition to a cloud platform [29]. Our approach can be integrated either as Software as a Service (SaaS) cloud variant. A general cloud-integrated solution open to 3rd party developers can be provided even as the Platform as a Service (PaaS) cloud variant.

Another research direction is to focus on the efficiency of the UI delivery process. Complex cloud platforms require an advanced multi-level UI delivery mechanism.

In our work about the distributed concern delivery described in [imp1] [wos1], we have shown the direction that could also be followed by the approach described in this dissertation thesis.

Currently, there is a trending development in the domain of the Internet of Things (IoT) [39]. We have already shown that it is possible to support solutions from this category using our platform. However, solutions for the seamless integration of such devices, as well as network discovery mechanisms, should be developed.



# Bibliography

- [1] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster. Uiml: an appliance-independent xml user interface language. *Computer Networks*, 31(11):1695–1708, 1999.
- [2] C. Abras, D. Maloney-Krichmar, and J. Preece. User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, 37(4):445–456, 2004.
- [3] M. Araki and K. Tachibana. Multimodal dialog description language for rapid system development. In *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*, pages 109–116. Association for Computational Linguistics, 2009.
- [4] J. Bardram and A. Friday. Ubiquitous computing systems. *Ubiquitous Computing Fundamentals*, pages 37–94, 2010.
- [5] J. Basek. Webove reseni pro podporu vyvoje v UIProtocolu (in Czech), Bachelor’s thesis, Czech Technical University in Prague, 2011.
- [6] E. Bernard. JSR 303: Bean validation, available from: <http://jcp.org/en/jsr/detail?id=303>, 2009, checked 2015-11-01.
- [7] S. Berti, F. Correani, G. Mori, F. Paternò, and C. Santoro. Teresa: a transformation-based environment for designing and developing multi-device interfaces. In *CHI’04 extended abstracts on Human factors in computing systems*, pages 793–794. ACM, 2004.
- [8] R. Biswas and E. Ort. The java persistence api-a simpler programming model for entity persistence. *Sun Microsystems, Inc., May*, 2006.
- [9] M. Blumendorf, G. Lehmann, and S. Albayrak. Bridging models and systems at runtime to build adaptive user interfaces. In *Proc. of the 2nd ACM SIGCHI*

- symposium on Engineering interactive computing systems*, pages 9–18. ACM, 2010.
- [10] J. Bryant and M. Jones. Responsive web design. In *Pro HTML5 Performance*, pages 37–49. Springer, 2012.
- [11] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
- [12] D. Carlson and A. Schrader. Dynamix: An open plug-and-play context framework for android. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 151–158. IEEE, 2012.
- [13] R. Cassino and M. Tucci. Developing usable web interfaces with the aid of automatic verification of their formal specification. *Journal of Visual Languages & Computing*, 22(2):140–149, 2011.
- [14] CBS Inc. Study: Number of smartphone users tops 1 billion., available from: <http://www.cbsnews.com/8301-205-162-57534583/study-number-of-smartphone-users-tops-1-billion/>, checked: 2013-06-10.
- [15] T. Cerny, V. Chalupa, and M. J. Donahoo. Towards smart user interface design. In *Info. Science and Applications (ICISA), 2012 Int. Conf. on*, pages 1–6. IEEE, 2012.
- [16] T. Cerny, M. J. Donahoo, and E. Song. Towards effective adaptive user interfaces design. In *Proc. of the 2013 Research in Applied Computation Symposium (RACS 2013)*, October 2013.
- [17] T. Cerny and E. Song. Uml-based enhanced rich form generation. In *Proc. of the 2011 ACM Symposium on Research in Applied Computation*, pages 192–199. ACM, 2011.
- [18] T. Cerny and E. Song. Model-driven rich form generation. *INFORMATION: An Int. Interdisciplinary Journal*, 15(7):2695–2714, July 2012.
- [19] J. Chattratichart and G. Lindgaard. A comparative evaluation of heuristic-based usability inspection methods. In *CHI'08 extended abstracts on Human factors in computing systems*, pages 2213–2220. ACM, 2008.

- [20] T. Clerckx, K. Luyten, and K. Coninx. The mapping problem back and forth: customizing dynamic models while preserving consistency. In *Proc. of the 3rd annual conf. on Task models and diagrams*, pages 33–42. ACM, 2004.
- [21] J. Conrad and J. Viescas. *Microsoft® Access® 2010 Inside Out*. O’Reilly Media, Inc., 2010.
- [22] V. Consortium et al. vcard—the electronic business card—version 2.1—specifications. *Internet citation,[Online] Sep*, 18:1–40, 1996.
- [23] D. Crockford. The application/json media type for javascript object notation (json), available from: <https://tools.ietf.org/html/rfc4627>, 2006, checked: 2015-10-10.
- [24] K. Czarnecki and U. W. Eisenecker. Components and generative programming (invited paper). In *Proc. of the 7th European software engineering conf., ESEC/FSE-7*, pages 2–19, London, UK, UK, 1999. Springer-Verlag.
- [25] M. Dalal and A. Ghoda. *XAML developer reference*. O’Reilly Media, Inc., 2011.
- [26] Danish Websearch Webzine. Wayfinding in hospitals, available from: <http://www.dcdr.dk/uk/menu/update/webzine/articles/wayfinding-in-hospitals>. [Online; accessed 24-April-2014].
- [27] L. DeMichiel. JSR 317: Java™ persistence API, version 2.0, available from: <http://jcp.org/en/jsr/detail?id=317>, November 2009, checked 2015-08-11.
- [28] S. Deterding, M. Sicart, L. Nacke, K. O’Hara, and D. Dixon. Gamification. using game-design elements in non-gaming contexts. In *CHI’11 Extended Abstracts on Human Factors in Computing Systems*, pages 2425–2428. ACM, 2011.
- [29] T. Dillon, C. Wu, and E. Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.
- [30] DIS, ISO. 9241-210: 2010. ergonomics of human system interaction-part 210: Human-centred design for interactive systems. *International Standardization Organization (ISO). Switzerland*, 2009.
- [31] B. Dumas, D. Lalanne, and R. Ingold. Description languages for multimodal interaction: a set of guidelines and its illustration with smuiml. *Journal on multimodal user interfaces*, 3(3):237–247, 2010.

- [32] J. Engel, C. Herdin, and C. Märtin. Evaluation of model-based user interface development approaches. In *Human-Computer Interaction. Theories, Methods, and Tools*, pages 295–307. Springer, 2014.
- [33] Eyedea recognition Ltd. Eyedentity – forensic software for face detection and recognition, available from: <http://www.eyedea.cz/eyedentity/>, 2015, checked: 2015-06-15.
- [34] J. Feiner and K. Andrews. Usability reporting with usabml. In *Human-Centered Software Engineering*, pages 342–351. Springer, 2012.
- [35] K. C. Feldt. *Programming Firefox: Building rich internet applications with XUL*. ” O’Reilly Media, Inc.”, 2007.
- [36] I. R. Forman and N. Forman. *Java Reflection in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.
- [37] K. Gajos, D. Weld, and J. Wobbrock. Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174(12-13):910–950, August 2010.
- [38] A. Gimblett and H. Thimbleby. Applying theorem discovery to automatically find and check usability heuristics. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 101–106. ACM, 2013.
- [39] D. Giusto, A. Iera, G. Morabito, and L. Atzori. *The internet of things: 20th Tyrrhenian workshop on digital communications*. Springer Science & Business Media, 2010.
- [40] A. M. Graziano and M. L. Raulin. *Research methods: A process of inquiry*. HarperCollins College Publishers, 1993.
- [41] L. Hakobyan, J. Lumsden, D. O’Sullivan, and H. Bartlett. Mobile assistive technologies for the visually impaired. *Survey of ophthalmology*, 58(6):513–528, 2013.
- [42] O. Hauptmann. UIP Applications Visual Editor. Master’s thesis, Czech Technical University in Prague, 2012.

- [43] IMS Global Learning Consortium, Inc. Ims learner information package specification, available from: <http://www.imsglobal.org/profiles/index.html>, 2008, checked 2015-11-01.
- [44] International Organization for Standardization. *ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability*. International Organization for Standardization, 1998.
- [45] ISO. IEC 24752: Information Technology—User Interfaces—Universal remote console—5 parts. *International Organization for Standardization*, 2008.
- [46] J. Jelinek and P. Slavik. Gui generation from annotated source code. In *Proceedings of the 3rd annual conference on Task models and diagrams*, pages 129–136. ACM, 2004.
- [47] N. Kaklanis, P. Biswas, Y. Mohamad, M. Gonzalez, M. Peissner, P. Langdon, D. Tzovaras, and C. Jung. Towards standardisation of user models for simulation and adaptation purposes. *Universal Access in the Information Society*, pages 1–28, 2014.
- [48] R. Kennard and J. Leaney. Towards a general purpose architecture for ui generation. *Journal of Systems and Software*, 83(10):1896–1906, 2010.
- [49] G. Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. *W3C Recommendation*, 2006.
- [50] M. Knappmeyer, N. Baker, S. Liaquat, and R. Tönjes. A context provisioning framework to support pervasive and ubiquitous applications. In *Smart Sensing and Context*, pages 93–106. Springer, 2009.
- [51] M. Knappmeyer, S. L. Kiani, C. Frà, B. Moltchanov, and N. Baker. Contextml: a light-weight context representation and context management schema. In *Wireless Pervasive Computing (ISWPC), 2010 5th IEEE International Symposium on*, pages 367–372. IEEE, 2010.
- [52] J. Kolb, P. Hübner, and M. Reichert. Automatically generating and updating user interface components in process-aware information systems. In *On the Move to Meaningful Internet Systems: OTM 2012*, pages 444–454. Springer, 2012.
- [53] G. Kramer. *Auditory display: Sonification, audification, and auditory interfaces*. Perseus Publishing, 1993.

- [54] S. Kurniawan and P. Zaphiris. Research-derived web design guidelines for older people. In *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, pages 129–135. ACM, 2005.
- [55] R. Laddad. *AspectJ in action: practical aspect-oriented programming*, volume 512. Manning Greenwich, 2003.
- [56] F. Lam and M. Longnecker. A modified wilcoxon rank sum test for paired data. *Biometrika*, 70(2):510–513, 1983.
- [57] W. Lechner and S. Baumann. Global navigation satellite systems. *Computers and Electronics in Agriculture*, 25(1):67–85, 2000.
- [58] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and V. López-Jaquero. Usixml: a language supporting multi-path development of user interfaces. In *Engineering human computer interaction and interactive systems*, pages 200–220. Springer, 2005.
- [59] V. López-Jaquero, F. Montero, and F. Real. Designing user interface adaptation rules with t: Xml. In *Proc. of the 14th int. conf. on Intelligent user interfaces*, pages 383–388. ACM, 2009.
- [60] K. Luyten, C. Vandervelpen, J. V. den Bergh, and K. Coninx. Context-sensitive user interfaces for ambient intelligent environments: Design, development and deployment.
- [61] M. Macík. *User Interface Generator, diploma thesis*. Czech Technical University in Prague, 2009.
- [62] I. S. MacKenzie. Fitts’ law as a research and design tool in human-computer interaction. *Human-computer interaction*, 7(1):91–139, 1992.
- [63] I. Maly and Z. Mikovec. Web applications usability testing with task model skeletons. In *Human-Centred Software Engineering*, pages 158–165. Springer, 2010.
- [64] D. L. McGuinness, F. Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.
- [65] B. McLaughlin. *Java & XML data binding*. ” O’Reilly Media, Inc.”, 2002.

- [66] T. Miaskiewicz and K. A. Kozar. Personas and user-centered design: How can personas benefit product design processes? *Design Studies*, 32(5):417–430, 2011.
- [67] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@run.time to support dynamic adaptation. *Computer*, 42(10):44–51, Oct. 2009.
- [68] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *Computers, IEEE Transactions on*, 100(9):917–922, 1977.
- [69] National Center for Health Statistics (US) and Council on Clinical Classifications and Commission on Professional and Hospital Activities and World Health Organization. *The International Classification of Diseases, 9th Revision, Clinical Modification: ICD. 9. CM*. Commission on Professional and Hospital Activities, 1978.
- [70] Netbeans.org. Swing gui builder (formerly project matisse), available from: <https://netbeans.org/features/java/swing.html>, checked: 2015-11-01.
- [71] J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 161–170. ACM, 2002.
- [72] J. Nichols, B. A. Myers, and B. Rothrock. Uniform: automatically generating consistent remote control user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 611–620. ACM, 2006.
- [73] G. Nicol, L. Wood, M. Champion, and S. Byrne. Document object model (dom) level 3 core specification. *W3C Working Draft*, 13:1–146, 2001.
- [74] J. Nielsen. *Usability engineering*. Morgan Kaufmann, 1993, ISBN 0125184069.
- [75] J. Nielsen. Ten usability heuristics, available from: <https://www.nngroup.com/articles/ten-usability-heuristics/>, 1995, checked 2016-01-10.
- [76] H. Okada and R. Fujioka. Automated methods for webpage usability & accessibility evaluations. *Advances in Human Computer Interaction, In-Tech Publishing, chapter 21*, pages 351–364, 2008.

- [77] D. R. Olsen Jr, S. Jefferies, T. Nielsen, W. Moyes, and P. Fredrickson. Cross-modal interaction using xweb. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 191–200. ACM, 2000.
- [78] E. J. O’Neil. Object/relational mapping 2008: hibernate and the entity data model (edm). In *Proc. of the 2008 ACM SIGMOD int. conf. on Management of data*, pages 1351–1356. ACM, 2008.
- [79] A. Parush and E. Morse. Industry usability reporting and the common industry format (ansi-ncits 354-2001). *The UPA (The Usability Professionals’ Association) Voice*, 5(1), 2003.
- [80] F. Paterno, C. Santoro, and L. D. Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(4):19, 2009.
- [81] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big’web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.
- [82] M. Peißner, T. Sellner, and D. Janssen. Myui individualization patterns for accessible and adaptive user interfaces. In *SMART 2012, The First International Conference on Smart Systems, Devices and Technologies*, pages 25–30, 2012.
- [83] J.-l. Perez-medina, S. Dupuy-chessa, and A. Front. A survey of model driven engineering tools for user interface design. In *In Proc. of 6th Int. workshop on Task Models and Diagrams (TAMODIA ’2007)*, pages 84–97, Berlin, 7-9 Nov. 2007. Springer.
- [84] M. Plessner and D. Lichtenberger. [fleXive] Content Repository. *JAVA Magazin*, page 22, 2010.
- [85] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. Icraft: A service framework for ubiquitous computing environments. In *UbiComp 2001: Ubiquitous Computing*, pages 56–75. Springer, 2001.
- [86] A. Puerta and J. Eisenstein. Ximl: a common representation for interaction data. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 214–215. ACM, 2002.



- [87] A. R. Puerta. A model-based interface development environment. *Software, IEEE*, 14(4):40–47, 1997.
- [88] L. Razmerita, A. Angehrn, and A. Maedche. Ontology-based user modeling for knowledge management systems. In *User Modeling 2003*, pages 213–217. Springer, 2003.
- [89] L. Richardson and S. Ruby. *RESTful web services*. ” O’Reilly Media, Inc.”, 2008.
- [90] R. D. Roberts, H. C. Beh, and L. Stankov. Hick’s law, competing-task performance, and intelligence. *Intelligence*, 12(2):111–130, 1988.
- [91] S. Salah and H. Sug. The effectiveness of rapid business application development using oracle forms. In *Advanced Information Management and Service (ICIPM), 2011 7th Int. Conf. on*, pages 33–37. IEEE, 2011.
- [92] J. Sauro and E. Kindlund. A method to standardize usability metrics into a single score. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–409. ACM, 2005.
- [93] M. Schlee and J. Vanderdonckt. Generative programming of graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, pages 403–406. ACM, 2004.
- [94] H.-J. Schönig and E. Geschwinde. *Mono kick start*. Sams Publishing, 2004.
- [95] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.
- [96] C. S. Sheets. level 2 (css2) specification. *W3C Recommendation*, 12, 1998.
- [97] Q. Z. Sheng and B. Benatallah. Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In *Mobile Business, 2005. ICMB 2005. International Conference on*, pages 206–212. IEEE, 2005.
- [98] V. Slovacek. Methods for efficient development of task-based applications. In *Human-Centred Software Engineering*, pages 206–213. Springer, 2010.

- [99] L. Smirek, G. Zimmermann, and D. Ziegler. Towards universally usable smart homes-how can myui, urc and openhab contribute to an adaptive user interface platform. In *IARIA Conference, Nice, France*, pages 29–38, 2014.
- [100] Software 602 a.s. Specifikace formátu 602xml formulářů (in czech), available from: [http://www.602.cz/datainc/602xml/technical/602XML\\_form\\_popis.pdf](http://www.602.cz/datainc/602xml/technical/602XML_form_popis.pdf), 2010, checked: 2015-10-10.
- [101] J.-S. Sottet, G. Calvary, J. Coutaz, and J.-M. Favre. A model-driven engineering approach for the usability of plastic user interfaces. In *Engineering Interactive Systems*, pages 140–157. Springer, 2008.
- [102] J.-S. Sottet, G. Calvary, and J.-M. Favre. Models at runtime for sustaining user interface plasticity. In *Models@ run. time workshop (in conjunction with MoDELS/UML 2006 conf.)*, 2006.
- [103] E. Standard. Final draft ETSI ES 202 746 V1. 1.1 (2009-12), 2009.
- [104] C. Stephanidis, A. Paramythis, D. Akoumianakis, and M. Sfyraakis. Self-adapting web-based systems: Towards universal accessibility. In *4th Workshop on User Interface For All, Stockholm, Sweden*, 1998.
- [105] P. A. Szekely, P. N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: the mastermind approach. In *EHCI*, pages 120–150. Citeseer, 1995.
- [106] V. Tran, M. Kolp, J. Vanderdonckt, Y. Wautelet, and S. Faulkner. Agent-based user interface generation from combined task, context and domain models. In *Task Models and Diagrams for User Interface Design*, pages 146–161. Springer, 2010.
- [107] A. Troelsen. *Pro C# 5.0 and the .NET 4.5 Framework*. Apress, 2012.
- [108] E. Upton and G. Halfacree. *Raspberry Pi user guide*. John Wiley & Sons, 2014.
- [109] A. Van Dam. Post-wimp user interfaces. *Communications of the ACM*, 40(2):63–67, 1997.
- [110] J. Vanderdonckt. Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments. *Proc. of XI Interacción*, pages 20–32, 2010.

- [111] M. Wertheimer. Laws of organization in perceptual forms. *A source book of Gestalt psychology*, 1938.
- [112] C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough method: A practitioner's guide. In *Usability inspection methods*, pages 105–140. John Wiley & Sons, Inc., 1994.
- [113] J. Wobbrock, S. Kane, K. Gajos, S. Harada, and J. Froehlich. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)*, 3(3):9, 2011.
- [114] World Health Organization. International classification of functioning, disability and health (icf), 2001.



# Refereed Publications in Journals with Impact Factor

- [imp1] T. Cerny, M. Macik, M. J. Donahoo, and J. Janousek. On distributed concern delivery in user interface design. *Computer Science and Information Systems*, 12(2):655–681, 2015, IF = 0.477 (2014). Ratio of author contribution: 60%, 15%, 15%, 10%.
- [imp2] M. Macik, T. Cerny, and P. Slavik. Context-sensitive, cross-platform user interface generation. *Journal on Multimodal User Interfaces*, 8(2):217–229, 2014, IF = 0.797 (2014). Ratio of author contribution: 50%, 40%, 10%.



# Refereed Publications in WoS

- [wos1] T. Cerny, M. Macik, M. J. Donahoo, and J. Janousek. Efficient description and cache performance in aspect-oriented user interface design. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1667–1676. IEEE, 2014, ratio of author contribution: 25%, 25%, 25%, 25%.
- [wos2] F. Hanzl, M. Macik, V. Slovacek, and M. Klima. Digital household user interfaces for seniors. In *AMIF*, pages 129–136, 2009, ratio of author contribution: 40%, 30%, 25%, 5%.
- [wos3] M. Klima, M. Macik, E. Urdaneta, C. Buiza, E. Carrasco, G. Epelde, and J. Alexandersson. User interfaces for the digital home on the basis of open industrial standards. In *AMIF*, pages 144–152. Citeseer, 2008.
- [wos4] M. Macik. Context model for ability-based automatic ui generation. In *Cognitive Infocommunications (CogInfoCom), 2012 IEEE 3rd Int. Conf. on*, pages 727–732. IEEE, 2012.
- [wos5] M. Macik, I. Maly, Z. Mikovec, and M. Urban. Addinpanel: Adaptive advertising interactive panel. In *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pages 375–380. IEEE, 2013, ratio of author contribution: 50%, 30%, 10%, 10%.





# Refereed Publications in Scopus

- [scopus1] K. Fixova, M. Macik, and Z. Mikovec. In-hospital navigation system for people with limited orientation. In *Cognitive Infocommunications (CogInfoCom), 2014 5th IEEE Conference on*, pages 125–130. IEEE, 2014, ratio of author contribution: 33%, 33%, 33%.
- [scopus2] M. Macik, T. Cerny, J. Basek, and P. Slavik. Platform-aware rich-form generation for adaptive systems through code-inspection. In *Human Factors in Computing and Informatics*, pages 768–784. Springer, 2013, ratio of author contribution: 55%, 25%, 10%, 10%.
- [scopus3] M. Macik, M. Klima, and P. Slavik. Ui generation for data visualisation in heterogenous environment. In *Advances in Visual Computing*, pages 647–658. Springer, 2011, ratio of author contribution: 50%, 25%, 25%.
- [scopus4] M. Macík, V. Slováček, and M. Klíma. User Interfaces for Intelligent Household. In *Ambient Intelligence European Conference, AmI 2009, Salzburg, Austria, November 18-21, 2009. Adjunct proceedings.*, pages 85–88, Salzburg, 2009, ratio of author contribution: 40%, 40%, 20%. ICT&S Center, University of Salzburg.
- [scopus5] M. Macik and A. J. Sporcka. Does user’s fatigue overweight the effect of practice. In *Proceedings of The Tenth IASTED International Conference on Software Engineering-SE 2011*, 2011, ratio of author contribution: 80%, 20%.



## Refereed Other Publications

- [other1] M. Macík. Centralized application logic for heterogeneous environment. In *POSTER 2010 - Proceedings of the 14th International Conference on Electrical Engineering*, pages 1–6, Prague, 2010. CTU, Faculty of Electrical Engineering.
- [other2] M. Macík, A. Kutíková, Z. Míkovec, and P. Slavík. GraFooSha: Food Sharing for Senior Users. In *6th IEEE Conference on Cognitive Infocommunications CogInfoCom 2015 PROCEEDINGS*, pages 267–272, Budapest, 2015. IEEE Hungary Section, University Obuda.
- [other3] M. Macík, E. Lorencova, Z. Míkovec, and O. Rakusan. Software architecture for a distributed in-hospital navigation system. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, pages 369–375. ACM, 2015.
- [other4] M. Macík, A. J. Sporcka, and P. Slavík. An initial study of effects of temporary disabilities. *ACM SIGACCESS Accessibility and Computing*, (103):3–13, 2012.
- [other5] P. Slavík, Z. Míkovec, M. Macík, and J. Balata. Problémy navigace seniorů v prostoru. In *Stárnutí 2014*, pages 120–132, Praha, 2014. Univerzita Karlova v Praze, 3. lékařská fakulta.



## Unrefereed Other Publications

- [otherun1] J. Balata, M. Macík, and Z. Míkovec. Context sensitive navigation in hospitals. 2013.
- [otherun2] J. Balata, Z. Míkovec, P. Slavík, and M. Macík. *Game Aspects in Collaborative Navigation of Blind Travelers*, pages –. IGI Global, Hershey, Pennsylvania, 2015.
- [otherun3] J. Balata, K. Prazakova, A. Kutikova, M. Macík, and Z. Mikovec. Quido: Arcade game with thermo-haptic feedback. 2013.
- [otherun4] J. Hušek and M. Macík. Multi-touch Table with Image Capturing. In *CESCG 2011*, pages 91–98, Vienna, 2011. Vienna University of Technology, Institute for Computer Aided Automation.
- [otherun5] M. Macík and V. Slováček. User interfaces for intelligent household remote controls. In *Proceedings of the 13th Central European Seminar on Computer Graphics*, pages 171–177, Vienna, 2009. Vienna University of Technology, Institute for Computer Aided Automation.
- [otherun6] V. Slováček, M. Macík, and M. Klíma. Development framework for pervasive computing applications. *ACM SIGACCESS Accessibility and Computing*, (95):17–29, 2009.



# Citations

- [citations1] E. C. Alonso. *Some Contributions to Smart Assistive Technologies*. PhD thesis, The University of Basque Country, 2015.
- [citations2] J. Arendsen. Exploring the design space for dynamic interfaces. *Institute for Computing and Information Sciences*, 2015.
- [citations3] P. Biswas, P. Langdon, J. Umadikar, S. Kittusami, and S. Prashant. How interface adaptation for physical impairment can help able bodied users in situational impairment. In *Inclusive Designing*, pages 49–58. Springer, 2014.
- [citations4] S. M. Butt, M. A. Majid, S. Marjudi, S. M. Butt, A. Onn, and M. M. Butt. Casi method for improving the usability of ids. *Science International*, 27(1), 2015.
- [citations5] T. Cerny, K. Cemus, M. J. Donahoo, and E. Song. Aspect-driven, data-reflective and context-aware user interfaces design. *ACM SIGAPP Applied Computing Review*, 13(4):53–66, 2013.
- [citations6] T. Cerny and M. J. Donahoo. On separation of platform-independent particles in user interfaces. *Cluster Computing*, pages 1–14, 2015.
- [citations7] T. Cerny and M. J. Donahoo. Separating out platform-independent particles of user interfaces. In *Information Science and Applications*, pages 941–948. Springer, 2015.
- [citations8] G. Epelde, X. Valencia, E. Carrasco, J. Posada, J. Abascal, U. Diaz-Orueta, I. Zinnikus, and C. Husodo-Schulz. Providing universally accessible interactive services through tv sets: implementation and validation with elderly users. *Multimedia tools and applications*, 67(2):497–528, 2013.

- [citations9] T. Nishimori and Y. Kuno. Join token-based event handling: a comprehensive framework for game programming. In *Software Language Engineering*, pages 119–138. Springer, 2012.
- [citations10] M. Pasealekua and M. Lardizabal. Providing universally accessible interactive services through tv sets: Implementation and validation with elderly users.
- [citations11] J. Porubän, M. Bačíková, S. Chodarev, and M. Nosál. Teaching pragmatic model-driven software development. *Computer Science and Information Systems*, 12(2):683–705, 2015.
- [citations12] G. Radhamani, K. Vanitha, and D. Rajeswari. Classification and handling of anonymity in pervasive middleware. *International Journal of Computer Applications*, 17(6):28–31, 2011.
- [citations13] M. Tomasek and T. Cerny. On web services ui in user interface generation in standalone applications. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, pages 363–368. ACM, 2015.
- [citations14] M. Trnka and T. Cerny. Context-aware role-based access control using security levels. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, pages 280–284. ACM, 2015.
- [citations15] E. Vildjiounaite, G. Gimel’farb, V. Kyllönen, and J. Peltola. Lightweight adaptation of classifiers to users and contexts: Trends of the emerging domain. *The Scientific World Journal*, 2015, 2015.
- Publication *Efficient description and cache performance in aspect-oriented user interface design* [wos1] was cited by [citations11].
  - Publication *Context-sensitive, cross-platform user interface generation* [imp2] was cited by [citations2, citations4, citations6, citations15, citations7, citations13, citations14].
  - Publication *Platform-Aware Rich-Form Generation for Adaptive Systems through Code-Inspection* [scopus2] was cited by [citations5, citations3].
  - Publication *UI generation for data visualisation in heterogenous environment* [scopus3] was cited by [citations5].



- Publication *Development framework for pervasive computing applications* [otherun6] was cited by [citations9, citations12].
- Publication *User Interfaces for the Digital Home on the basis of Open Industrial Standards*. [wos3] was cited by [citations8, citations10, citations1].



# Appendix A

## Tools for development support

This appendix shows examples of development support tools that have been created to simplify the work with the *Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)* and the corresponding *UIP Platform*. Here, we show two tools that aim to support developers dealing with it. Firstly, in section A.1, we describe the *UIP Web Portal* – a web solution to support the *UIP Platform* development. Secondly, in section A.2, we show further details about the *UIP Visual Editor*.

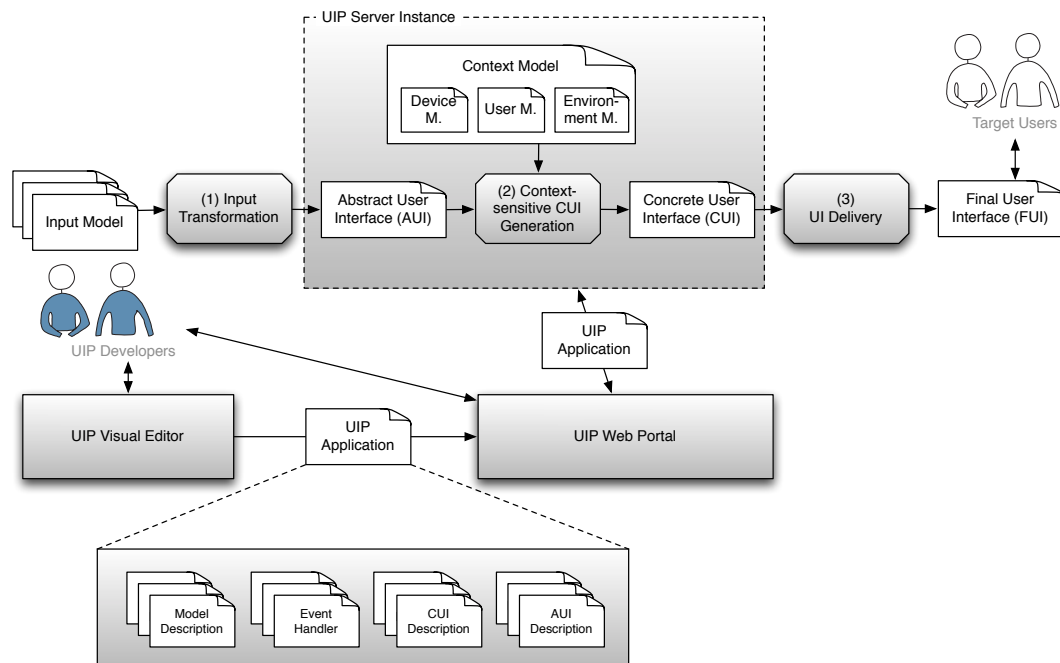


Figure A.1: Scope of the *UIP Development Support Tools* in the framework of Context-sensitive Automatic Concrete User Interface Generation Pipeline (UiGE Pipeline)

Figure A.1 depicts the relationship between tools for UIP development support and the UiGE Pipeline. The estimated user audience of our tools – *UIP developers* are expected to have experience with computer programming (preferably Java, Javascript, .NET) and also to be familiar with the basic concepts behind the *UIP Platform*. Note that the *UIP Application* format, see section 3.3.2 serves as the exchange format between particular development support tools and the *UIP Server*.

## A.1 UIP Web Portal

In this section, we describe the *UIP Web Portal*, which aims to support developers that deal with *UIP Platform* development. This tool has been created in the framework of Bachelor's Thesis [5] advised by this dissertation thesis author.

The original implementation of the *UIP Platform* can serve as runtime for one *UIP Application* on one *UIP Server* instance at the same time. This is enough for experimental development and as a runtime for various one-purpose applications. The runtime configuration of the original *UIP Platform* is also rather complicated for new developers dealing with it. This induces the motivation to create a solution that enables easier interaction with the *UIP Infrastructure*. Apart from the current *UIP* developers, the new tool should also enable new *UIP* developers to get familiar with the *UIP Platform* easier.

Following list summarizes the most important functional requirements to the *UIP Web Portal*:

- *UIP Application management*: The *UIP Web Portal* should provide an easy way how to deploy *UIP applications* and to provide tools how to control corresponding *UIP application* deployments.
- *UIP Server instance management*: The *UIP Web Portal* should simplify management of *UIP Server instances* that serve as a runtime for deployed *UIP Applications*.
- *Content Management System*: The *UIP Web Portal* should integrate a Content Management System to enable publishing news articles related to the *UIP Platform*.
- *User management*: The *UIP Web Portal* should enable advanced user management supporting different user roles. Access rights to different *UIP Appli-*

ctions deployed in the *UIP Web Portal* as well as right to manage other *UIP Web Portal* features should be defined.

The screenshot displays the UIPortal web interface. The header includes the logo '<UIPortal />' and the tagline 'WEB SOLUTION FOR UIPROTOCOL DEVELOPMENT SUPPORT'. The navigation menu contains 'HOME', 'UIP APPLICATIONS', 'ARTICLES', 'FILES', and 'SERVER MANAGEMENT'. The main content area is titled 'UiTV Color 1 (for application: UiTV Color)'. It shows metadata such as 'Created by SUPERVISOR', 'Creation time 2011-04-30 10:34:47.453', 'Modified by SUPERVISOR', 'Modification time 2011-04-30 10:34:50.047', 'Application ID uitvcolor', 'Instance ID uitvcolor1', and 'Visibility visible'. Below this, there are 'Edit', 'Delete', and 'Select instance' buttons. A section titled 'Instance status on UIPServers:' contains a table with the following data:

Server Address	XML port HTTP port	Clients on server	Instances on server	Status	Clients on instance	Instance
server1 192.168.56.1	5678 4567	0	0 from 20	Unloaded	0 from 40	

Below the table, a message states: 'Instance status data are valid for page load. For newest data reload page.' There is also a 'Comments' section with a 'New comment' form containing a 'Comment text' input field and a 'Create comment' button. The footer contains copyright information: '© Jindrich Batek, FEE CTU, 2011. Design by Free CSS Templates. Powered by JaviZEE and fleXive framework.'

Figure A.2: UIP Portal – UIP Application management

The above mentioned functional requirements have been addressed by the *UIP Web Portal* implementation. This web solution is based on [fleXive] framework [84]. Figure A.2 shows an example of *UIP Web Portal* UI. This particular screen depicts the *UIP Application* management function. Notice the information about status of related *UIP Server* instances. Part of the *UIP Web Portal* solution is *UIP Server* implementation in Java, that enables running *UIP Applications* in multiple instances. This solution is even capable of running *UIP Server* instances simultaneously.

It is subject of the future work to perform usability evaluation of the *UIP Web Portal*. Part of the documentation described in [5] is a user guide. The *UIP Web Portal* solution could also serve as the basis for *UIP Platform* transition into the cloud.

## A.2 UIP Visual Editor

The basic information about the *UIP Visual Editor* has been already described in section 4.2. The *UIP Visual Editor* has been developed as part of a Master's Thesis, see [42] (advised by this dissertation thesis author). The aim of this development tool was to simplify work with *UIP Applications*.

*UIP Visual Editor* development was based principles of the User Centered Design (UCD) [2]. The target user audience are developers that are familiar with the basic concepts behind the *UIP Platform* like the concepts of Abstract User Interface (AUI), Concrete User Interface (CUI) and CM. We also expect our users to be familiar with the Context-sensitive Automatic Concrete User Interface Generation Pipeline. On the other hand, target users (developers) do not need to deal with implementation details behind the *UIP Platform*.

Figure A.3 shows low-fidelity prototype of the *UIP Visual Editor*. For purposes of AUI creation and editing, the *UIP Visual Editor* uses graphical representations of elements that shows their abstract manner. Graphical representation of *AUI Elements* was created to clearly represent an abstract element (e.g. one-from-N selection) but not to evoke a corresponding CUI representation (e.g. combo-box). Figure A.4 illustrates the evolution of the low-fidelity prototype of the *UIP Visual Editor*, in this case a window for editing element properties.

Figure A.5 shows an example of an AUI visualized by the AUI visual editor. The representation of the AUI elements depends on the context, and is a result of

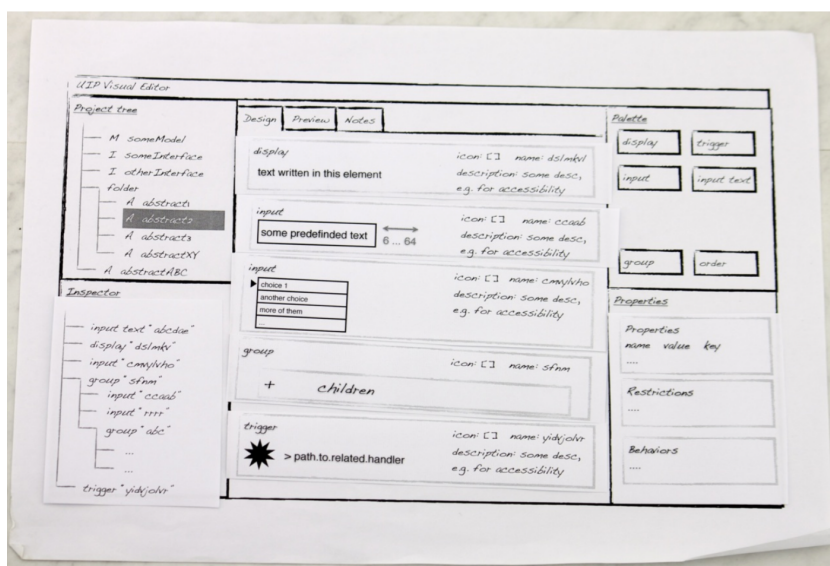


Figure A.3: Low-Fidelity prototype of *UIP Visual Editor*, from [42]



Figure A.4: Evolution of property window of the Low-Fidelity prototype of the *UIP Visual Editor*, from [42]

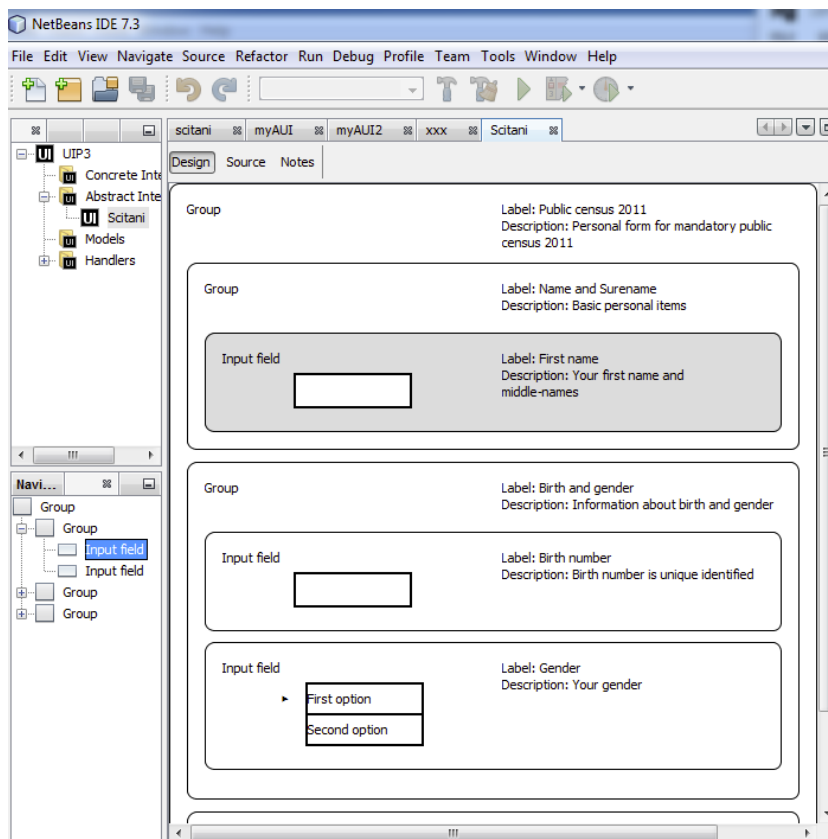


Figure A.5: *UIP Visual Editor* – editing AUI

the UI generation. Note that the visualization of individual elements is rendered in a context-independent way. AUI elements are very general – display, input and trigger. The final representation of these elements can be affected by restriction properties, e.g. the set of possible values. For example, there are only two possible values for gender – male and female. Note the different visualization of such an abstract element in the AUI visual editor.

Figure A.6 shows deployment of a *UIP Application* from *UIP Visual Editor* using the above described *UIP Web Portal*. The automated deployment should further simplify the *UIP Platform* development and bring synergy of the combination of the *UIP Visual Editor* and the *UIP Web Portal*.

### A.2.1 UIP Visual Editor Evaluation

The *UIP Visual Editor* has been evaluated using the cognitive walkthrough method [112]. The evaluation covered eleven basic editor use-cases, including *Creation of a UIP Project*, *Reordering of Elements in a CUI*, *Binding Element Properties*, *Setup of the Label of an AUI Element*, and *Deploy to the UIP Portal*.

As an outcome, the evaluation discovered potentially severe issues regarding developer with little familiarity with the *UIP Platform*. The level of abstraction of *AUI Elements* could still be rather low to isolate developers from the actual source-code representation in the *UIP AUI Format*.

It is subject of the future work to preform complex usability evaluation of both, the *UIP Visual Editor* and *UIP Portal* using the corresponding target user audience.

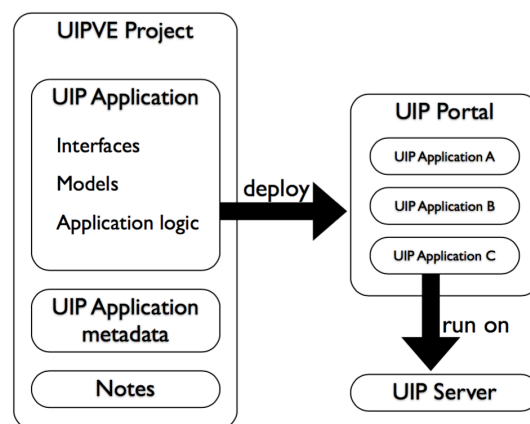


Figure A.6: Deployment of a *UIP Application* from the *UIP Visual Editor*, from [42]



# Appendix B

## Source Code Examples

### B.1 UIP AUI Example

```
1 <XML header>
2 <interfaces>
3   <interface class="oz.ctu.hvac_example.root">
4     <label>
5       <property name="title" value="Home_Heating_Control">
6     </label>
7     <container>
8       <label>
9         <property name="title" value="Select_Room">
10      </label>
11      <element>
12        <label>
13          <property name="title" value="OK">
14        </label>
15        <behaviors>
16          <behavior trigger="action" action="home.temperature.confirm" />
17        </behaviors>
18      </element>
19    </container>
20    <container>
21      <label>
22        <property name="title" value="Set_Room_Temperature">
23      </label>
24      <element class="public.input">
25        <label>
26          <property name="title" value="Temperature">
27        </label>
28      </element>
29      <element class="public.trigger">
30        <label>
31          <property name="title" value="OK">
32        </label>
33        <behaviors>
34          <behavior trigger="action" action="home.temperature.confirm" />
35        </behaviors>
36      </element>
37    </container>
38  </interface>
39 </interfaces>
```

Listing B.1: Structure of AUI

## B.2 Context model properties

Following tables B.1 and B.2 illustrate Context Model (CM) parameters that were used for realization of examples described in chapters 7 - 9. Sets of parameters represented in the tables bellow were inspired by [103] (User Model). During the development, some parameters (e.g. `key_size` – element spacing) were also added to meet requirements of a particular application use-case.

### B.2.1 Device model

Property	Example value	Description
<code>id</code>	<code>cz.ctu.uip.client</code> <code>.wpf.desktop</code>	<i>UIP Client</i> identifier.
<code>line_width</code>	2 px	Minimal absolute line width a typical user can recognize.
<code>target_size</code>	150x50 px	Minimal absolute target size a typical user can use.
<code>font_height</code>	12 px	Minimal absolute font size a typical user can recognize.
<code>sound_volume</code>	60	Minimal absolute sound volume a typical user can recognize.
<code>speech_volume</code>	70	Minimal absolute speech volume a typical user can recognize.
<code>key_size</code>	8 px	Minimal absolute key size – minimal spacing between UI elements.
<code>contrast</code>	0.5	Minimal absolute contrast a typical user can use (in client device units).
<code>brightness</code>	0.8	Minimal absolute brightness a typical user can use (in client device units).
<code>key_press_time</code>	0.2 s	Minimal duration of key press that is recognized as positive input (for a typical user).
<code>maximum_contrast</code>	1.0	Maximum contrast value the <i>UIP Client</i> device can provide.
<code>maximum_brightness</code>	1.0	Maximum brightness value the <i>UIP Client</i> device can provide.

maximum_volume	255	Maximum volume value the <i>UIP Client</i> device can provide.
minimal_user_height	0 cm	Minimal user height to interact with the UIP client device.
v screen_width	1024 px	Current screen width.
screen_height	768 px	Current screen height.
screen_dpi	150	Current screen dpi.
supported_elements	public.input.text, public.trigger	List of supported UIP CUI elements.
is_touchscreen	false	Specifies whether the device is touchscreen.
is_multitouch	false	Specifies whether the device support multi-touch.
has_keyboard	false	Specifies whether the device physical keyboard.
has_mouse	true	Specifies whether the device physical mouse or similar input device.

Table B.1: Device model properties

### B.2.2 User model

Property	Example value	Description
id	u003	User unique identifier.
line_width	1.2	Minimal relative line width the related user can recognize.
target_size	1.5	Minimal relative target size the related user can use.
font_height	1.5	Minimal relative font size the related user can recognize.
sound_volume	1.0	Minimal relative sound volume the related user can recognize.
speech_volume	1.0	Minimal relative speech volume the related user can recognize.

key_size	1.0	Minimal relative key size – minimal spacing between UI elements.
contrast	1.2	Minimal relative contrast the related user can use (in client device units).
brightness	0.8	Minimal relative brightness the related user can use (in client device units).
key_press_time	2.3	Minimal relative duration of key press that is recognized as positive input.
one_hand	false	Represents whether can only use one hand for interaction.
no_hand	false	Represents whether can not use any hand for interaction.
blind	false	Represents whether user is blind.
body_height	168 cm	Represents user height.
in_wheelchair	false	Refers to whether user is in wheelchair.
info_text	0.9	Text representation quotient (0.0 - impossible, 1 - no problem).
info_picture	0.7	Graphics representation quotient (0.0 - impossible, 1 - no problem).
info_colors	0.7	Color representation quotient (0.0 - impossible, 1 - no problem).
info_sound	0.5	Sound representation quotient (0.0 - impossible, 1 - no problem).
info_speech	0.5	Speech representation quotient (0.0 - impossible, 1 - no problem).
info_simple_haptic	0.5	Haptic representation quotient (0.0 - impossible, 1 - no problem).
info_braille_code	0.0	Brail code representation quotient (0.0 - impossible, 1 - no problem).
ic_quocient	0.7	Information complexity quotient. Refers to how complex information can user understand.

language.cs	1.0	Language knowledge quotient – Czech language (0.0 - no knowledge, 1.0 - perfect knowledge).
language.en	0.40	Language knowledge quotient – English language (0.0 - no knowledge, 1.0 - perfect knowledge).
language.de	0.25	Language knowledge quotient – German language (0.0 - no knowledge, 1.0 - perfect knowledge).
language.fr	0.20	Language knowledge quotient – French language (0.0 - no knowledge, 1.0 - perfect knowledge).

Table B.2: User model properties