

Open Research Online

The Open University's repository of research publications
and other research outputs

Engineering Adaptive Model-Driven User Interfaces for Enterprise Applications

Thesis

How to cite:

Akiki, Pierre A. (2014). Engineering Adaptive Model-Driven User Interfaces for Enterprise Applications. PhD thesis The Open University.

For guidance on citations see [FAQs](#).

© 2014 Pierre Akiki

Version: Version of Record

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

oro.open.ac.uk

Department of Computing and Communications
Faculty of Mathematics, Computing and Technology
The Open University

**Engineering Adaptive
Model-Driven User Interfaces
for Enterprise Applications**

Pierre A. Akiki

**A thesis submitted in fulfillment of the requirements
for the degree of
Doctor of Philosophy in Computing**

**United Kingdom
May 2014**

To my family:

My father Antoine and my mother Randa

My brother Paul and my sister Patricia

“Work is love made visible. And if you cannot work with love but only with distaste, it is better that you should leave your work and sit at the gate of the temple and take alms of those who work with joy.”

—Gibran Khalil Gibran, *The Prophet*

This thesis was created with love, and the joy of making a novel contribution to the world of knowledge.

“The Cedars of Lebanon are the most famous natural monuments in the universe. Religion, poetry and history have all celebrated them because of the reputation for magnificence and holiness that these prodigies of vegetation have enjoyed since the earliest antiquity... These ancient witnesses of past ages know history better than does history itself ...”

—Alphonse de Lamartine

Part of the contribution presented in this thesis is labelled with the word “Cedar”, to honor the legacy of the Cedars of Lebanon.

Abstract

Enterprise applications such as enterprise resource planning systems have numerous complex user interfaces (UIs). Usability problems plague these UIs because they are offered as a generic off-the-shelf solution to end-users with diverse needs in terms of their required features and layout preferences. Adaptive UIs can help in improving usability by tailoring the features and layout based on the context-of-use. The model-driven UI development approach offers the possibility of applying different types of adaptations on the various UI levels of abstraction. This approach forms the basis for many works researching the development of adaptive UIs. Yet, several gaps were identified in the state-of-the-art adaptive model-driven UI development systems. To fill these gaps, this thesis presents an approach that offers the following novel contributions:

- The **Cedar Architecture** serves as a reference for developing adaptive model-driven enterprise application user interfaces.
- **Role-Based User Interface Simplification** (RBUIS) is a mechanism for improving usability through adaptive behavior, by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use.
- **Cedar Studio** is an integrated development environment, which provides tool support for building adaptive model-driven enterprise application UIs using RBUIS based on the Cedar Architecture.

The contributions were evaluated from the technical and human perspectives. Several metrics were established and applied to measure the technical characteristics of the proposed approach after integrating it into an open-source enterprise application. Additional insights about the approach were obtained through the opinions of industry experts and data from real-life projects. Usability studies showed the approach's ability to significantly improve usability in terms of end-user efficiency, effectiveness and satisfaction.

Keywords: Adaptive UIs, Model-Driven Engineering, Enterprise Applications

Author's Declaration

The work presented in this thesis is an original contribution of the author. Parts of this thesis were published in the following peer-reviewed papers:

Journal Article

- P. Akiki, A. Bandara, Y. Yu. **Adaptive Model-Driven User Interface Development Systems**, ACM Computing Surveys, 47(1), ACM (2015)

Chapter 2

(Akiki et al. 2015)

Conference Proceedings

- P. Akiki, A. Bandara, Y. Yu. **Integrating Adaptive User Interface Capabilities in Enterprise Applications**. Proceedings of the 36th International Conference on Software Engineering (ICSE), IEEE/ACM (2014), Hyderabad, India

Chapter 7

(Akiki et al. 2014)

- P. Akiki, A. Bandara, Y. Yu. **RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior**. Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), ACM (2013), London, U.K. **Best Paper Award**

Chapter 5

(Akiki et al. 2013d)

- P. Akiki, A. Bandara, Y. Yu. **Cedar Studio: An IDE Supporting Adaptive Model-Driven User Interfaces for Enterprise Applications**. Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), ACM (2013), London, U.K.

Chapter 6

(Akiki et al. 2013a)

- P. Akiki, A. Bandara, Y. Yu. **Crowdsourcing User Interface Adaptations for Minimizing the Bloat in Enterprise Applications.** Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), ACM (2013), London, U.K.

Chapter 8

(Akiki et al. 2013b)

- P. Akiki, A. Bandara, Y. Yu. **Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications.** Proceedings of the 14th International Conference on Enterprise Information Systems (ICEIS), SciTePress (2012), Wroclaw, Poland

Chapter 4

(Akiki et al. 2012)

Book Chapter

- P. Akiki. **Devising a New Model Driven Framework for Developing GUI for Enterprise Applications.** Information Systems Development – Towards a Service Provision Society, Chapter 28, Springer (2010).

An early investigation of the research topic (before the PhD)

(Akiki 2010)

Doctoral Consortium

- P. Akiki, A. Bandara, Y. Yu. **Engineering Adaptive User Interfaces for Enterprise Applications.** Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Comp. Systems (EICS), ACM (2013), London, U.K.

Chapter 3

(Akiki 2013)

Workshop Proceedings

- P. Akiki, A. Bandara, Y. Yu. **Preserving Designer Input on Concrete User Interfaces Using Constraints While Maintaining Adaptive Behavior.** Proceedings of the 2nd Workshop on Context-Aware Adaptation of Service Front-Ends (CASFE), CEUR (2013), London, U.K.

Chapter 8

(Akiki et al. 2013c)

Acknowledgments

I would like to thank my supervisors Arosha Bandara and Yijun Yu for their invaluable support and guidance. Their comments and advice helped in improving the quality of this work.

I am grateful to: Sheep Dalton, Bashar Nuseibeh, Marian Petre, and Helen Sharp, for taking the time to offer me comments on some of my papers. I would also like to thank my mini-viva examiners Sheep Dalton and Michel Wermelinger for their comments.

I owe thanks to Bashar Nuseibeh and my supervisors for involving me in the ASAP project activities such as: the OU-NII-Lero workshop and the NII Shonan meeting. I am also grateful for the funding, which I got from the ERC Advanced Grant 291652, to support some of my trips to academic events.

A thank you goes to Robin Laney and all those who contributed to the postgraduate student forum and conference, which serve as great means for orienting and supporting postgraduate students during their research.

I am grateful to all the people who dedicated time to participate in my studies. Your contribution was vital for the completion of this work.

Many thanks go to my family and The Open University community for their support throughout the PhD process. A special thank you goes to the students of the Computing and Communications department for the interesting discussions and the fun that we had together.

Finally, I would like to thank The Open University and its Computing and Communications Department for offering me funding and the opportunity to pursue a PhD degree in a friendly and supportive environment.

Table of Contents

ABSTRACT	V
AUTHOR'S DECLARATION	VII
ACKNOWLEDGMENTS	IX
TABLE OF CONTENTS	XI
LIST OF FIGURES.....	XVII
LIST OF TABLES	XXI
LIST OF EQUATIONS	XXIII
LIST OF CODE LISTINGS	XXV
LIST OF ABBREVIATIONS	XXVII
GLOSSARY.....	XXIX

1 Introduction: Enterprise Application Usability Problems and Solutions 1

1.1 Definitions: Enterprise Applications and Adaptive UIs	2
1.1.1 What is an Enterprise?	2
1.1.2 What are Enterprise Applications?.....	2
1.1.3 The Characteristics of Enterprise Application User Interfaces.....	3
1.1.4 What are Adaptive User Interfaces?.....	3
1.2 Problem Definition.....	4
1.3 Research Motivation	5
1.3.1 Enterprise Application Revenues and Adoption Rate	5
1.3.2 Usability Problems Reported in the Literature.....	6
1.3.3 The Impact of Usability Problems on ERP Implementation Success .	7
1.3.4 UI Adaptation can Improve Enterprise Application Usability	9
1.4 Research Objectives	13
1.5 Thesis Organization.....	14

2 Literature Review: Adaptive Model-Driven User Interface Development Systems 17

2.1 Introduction	17
2.2 Approaches To User Interface Development and adaptation.....	21
2.2.1 Traditional Development : Programming, Event, and Markup Languages.....	22

2.2.2 Window Managers and Widget Toolkits.....	24
2.2.3 Model-Driven Engineering.....	25
2.2.4 Summary	27
2.3 Background.....	28
2.3.1 Model-Based User Interface Development.....	29
2.3.1.1 First and Second Generation MBUID Systems	29
2.3.1.2 Third Generation MBUID Systems.....	31
2.3.2 Reference Architectures for Adaptive Systems	33
2.3.3 Architectural Patterns and the Separation of Concerns.....	34
2.4 Criteria for Evaluating Adaptive Model-Driven User Interface Development Systems	36
2.5 Reference Architectures for Adaptive User Interfaces	42
2.5.1 Review	42
2.5.2 Summary of the Review	44
2.6 Techniques for Devising Adaptive Model-Driven UIs	45
2.6.1 Feature-Set Adaptation Techniques.....	45
2.6.1.1 Review	46
2.6.1.2 Summary of the Review.....	47
2.6.2 Layout Optimization Techniques	48
2.6.2.1 Review	48
2.6.2.2 Summary of the Review.....	53
2.7 Tools Supporting Adaptive Model-Driven UI Development.....	54
2.7.1 Review	55
2.7.2 Summary of the Review.....	60
2.8 Chapter Summary	61

3 Research Design: Research Questions, Hypotheses, and Methods..... 65

3.1 Research Questions	65
3.1.1 Technical Contribution Research Questions	66
3.1.2 Evaluation Research Questions.....	68
3.1.2.1 Software Engineering Perspective	68
3.1.2.2 Human-Computer Interaction Perspective	68
3.2 Hypotheses.....	69
3.3 Research Questions for Future Work (Partially Addressed)	70
3.4 Research Methods.....	70
3.4.1 Engineering Techniques	70
3.4.2 Empirical Methods.....	71
3.4.2.1 Surveys.....	71
3.4.2.2 Worked Examples and Software Metrics	71
3.4.2.3 Controlled Experiments.....	72
3.4.2.4 Interviews	72

3.5 Building on Existing Research and Technologies	72
3.6 Chapter Summary	73

4 The Cedar Architecture: A Reference for Developing Adaptive Model-Driven Enterprise Application User Interfaces 75

4.1 Introduction	75
4.2 The Cedar Architecture	77
4.2.1 Using Interpreted Runtime Models	78
4.2.2 Layers Comprising the Cedar Architecture.....	79
4.2.2.1 Client Components Layer	79
4.2.2.2 Decision Components Layer	81
4.2.2.3 Adaptation Components Layer	81
4.2.2.4 Adaptive Behavior and UI Models Layer	82
4.2.3 Adaptive Behavior Data: How Should the UI be Adapted?	82
4.2.4 Adaptation Procedure	83
4.3 General-Purpose Meta-Model.....	85
4.3.1 Multi-Aspect Adaptive User Interfaces Meta-Model.....	85
4.3.2 User Interface Levels of Abstraction Meta-Model.....	87
4.3.2.1 Task Models	88
4.3.2.2 Abstract User Interface Models	88
4.3.2.3 Concrete User Interface Models	89
4.4 Evaluating the Performance of Interpreted Runtime Models	91
4.5 Chapter Summary	92

5 RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior 95

5.1 Introduction	95
5.2 Role-Based User Interface Simplification (RBUIS)	98
5.3 Minimizing the User Interface's Feature-Set	99
5.3.1 Feature-Set Minimization with RBUIS.....	100
5.3.2 Less Time Consuming Access Rights Allocation	100
5.3.3 Applying RBUIS to Task Models at Runtime.....	101
5.3.4 Model Checking Using SQL	103
5.3.5 Feature-Set Minimization Example	104
5.4 Optimizing the User Interface's Layout.....	106
5.4.1 Layout Optimization with RBUIS and Workflows.....	107
5.4.2 Applying RBUIS at Runtime using Workflows	108
5.4.3 Layout Optimization Example.....	108
5.5 End-User Feedback for Refining the Adaptations	111
5.6 Managing Trade-Offs in Multi-Aspect Adaptive UIs.....	112

5.6.1 Trade-Off Analysis Technique	113
5.6.2 Multi-Aspect Trade-Off Analysis Example.....	120
5.7 Refitting the UI's Layout After Adaptation	122
5.8 Discussion	124
5.9 Chapter Summary	127

6 Cedar Studio: An IDE Supporting the Development of Adaptive Model-Driven User Interfaces for Enterprise Applications 129

6.1 Introduction	129
6.2 Design Tools for User Interface Models	132
6.2.1 Task Models	132
6.2.2 Abstract User Interface Models	136
6.2.3 Concrete User Interface Models	137
6.3 Design Tool for Adaptive Behavior Workflows	138
6.4 Design Tool for Goal Models.....	140
6.5 Testing the Adapted User Interfaces	141
6.6 Assessing Cedar Studio	143
6.7 Chapter Summary	146

7 Evaluating the Contributions from the Technical and Human Perspectives 147

7.1 Introduction	147
7.2 Evaluation Based on Technical Metrics	149
7.2.1 How Does the State-of-the-Art Integrate in Existing Systems?	150
7.2.2 Integrating RBUIS in OFBiz	152
7.2.2.1 Integration Based on the Cedar Architecture	153
7.2.2.2 The Technique for Integrating RBUIS in OFBiz	154
7.2.3 Metric-Based Evaluation	157
7.2.3.1 Reverse-Engineering the User Interfaces	158
7.2.3.2 Integrating the Adaptive UI Capabilities	160
7.2.3.3 Level of Decoupling.....	164
7.2.3.4 Runtime Performance.....	166
7.2.4 Discussing the Results of the Technical Evaluation	169
7.2.4.1 Discussion	169
7.2.4.2 Threats to Validity and Limitations.....	170
7.3 Evaluation Based on Industrial Expertise and Data	171
7.3.1 Inquiring about our Approach's Generality and Flexibility.....	171
7.3.2 Importance of the Feedback Loop in the UI Adaptation Process ...	172
7.3.3 Discussing the Results of Interviewing the Industry Expert	175
7.3.3.1 Discussion	175

7.3.3.2 Threats to Validity	176
7.4 Evaluation Based on Usability Studies.....	176
7.4.1 Online Study.....	177
7.4.1.1 Participant Recruitment and Demographics.....	179
7.4.1.2 Task Allocated to Participants.....	180
7.4.1.3 Results.....	181
7.4.2 Lab-Based Study	182
7.4.2.1 Participant Recruitment and Demographics.....	188
7.4.2.2 Tasks Allocated to Participants.....	188
7.4.2.3 Satisfaction and Efficiency Results	190
7.4.2.4 Effectiveness Results	194
7.4.2.5 Eye-Tracking Results.....	196
7.4.2.6 Additional Insights	199
7.4.3 Discussing the Results of the Usability Studies.....	200
7.4.3.1 Discussion.....	200
7.4.3.2 Threats to Validity	202
7.5 Chapter Summary	203
8 Conclusions and Future Work	205
8.1 Contributions	205
8.2 Future Work	208
8.2.1 Preserving Designer Input on the User Interface	208
8.2.2 Empowering New Design Participants.....	209
8.2.3 Applying Simplification to Multiple Related User Interfaces.....	209
8.3 Final Thoughts.....	210
BIBLIOGRAPHY	211
APPENDICES	229
A Algorithms	231
A.1 Feature-Set Minimization Algorithm	231
A.2 Layout Optimization Algorithm	236
A.3 Conflict Checking Based on Temporal Constraints	237
B Questionnaires.....	239
B.1 Demographics Questions Used in Usability Studies.....	239
B.2 System Usability Scale (SUS).....	242
B.3 Microsoft Product Reaction Cards	243

List of Figures

Figure 1.1: Enterprise Application Software Worldwide Revenue Forecast Comparison by Segment between the Years 2011 and 2016 (Gartner 2013).....	6
Figure 1.2: Example Question from Survey on Variations in Preferences among End-Users with Different Levels of Computer Literacy	11
Figure 1.3: Variance in UI Preferences among the Different Computer Literacy Groups.....	12
Figure 2.1: Self-* Properties of Adaptive User Interfaces	18
Figure 3.1: An Early Prototype of Our IDE (Akiki 2010)	67
Figure 3.2: Overview of the Research Steps	73
Figure 4.1: The Cedar Architecture	77
Figure 4.2: User Interface Adaptation Procedure.....	84
Figure 4.3: Meta-Model for Multi-Aspect Adaptive User Interfaces	86
Figure 4.4: Meta-Model for User Interface Levels of Abstraction	87
Figure 4.5: Task Model for Customer Maintenance UI	88
Figure 4.6: Abstract UI Model for Customer Maintenance UI.....	89
Figure 4.7: Concrete UI Model for Graphical Customer Maintenance UI	90
Figure 4.8: Performance Comparison between a UI based on Interpreted Runtime Models and a Code-Based One	92
Figure 5.1: Cedar Architecture Components that are realized by RBUIS.....	97
Figure 5.2: Meta-Model of Applying RBUIS to the Task Model.....	99
Figure 5.3: Simplified Customer Maintenance Task Model	104
Figure 5.4: Feature-Set Minimization of Customer Maintenance UI	105
Figure 5.5: Meta-Model for Applying RBUIS to the CUI using Workflows	106

Figure 5.6: Layout Optimization Adaptive Behavior Workflow	109
Figure 5.7: Optimized Layout of Customer FUI.....	110
Figure 5.8: User Feedback Interface Showing Simplification Operations.....	111
Figure 5.9: Business Partners Mobile User Interface based on the SAP Business One ERP Mobile Application.....	113
Figure 5.10: Multi-Aspect/Factor Adaptive UI Goal Model Example	114
Figure 5.11: Pareto Front for Multi-Aspect Trade-Off	117
Figure 5.12: Excerpt from a Bank Account Maintenance UI Example with Layout Adaptation.....	122
Figure 5.13: UI Refitting Example Using Relative Positioning	123
Figure 6.1: Parts of Cedar Architecture and RBUIS Supported by Cedar Studio	130
Figure 6.2: Task Model Design Tool.....	132
Figure 6.3: Visual Role Allocation.....	133
Figure 6.4: RBUIS Rules Code Editor.....	134
Figure 6.5: Model-Checking Constraints Code Editor.....	135
Figure 6.6: Mapping Task Model to AUI	135
Figure 6.7: Abstract User Interface Design Tool	136
Figure 6.8: Mapping AUI to CUI	137
Figure 6.9: Concrete User Interface Design Tool	138
Figure 6.10: Adaptive Behavior Workflow Design Tool.....	139
Figure 6.11: Dynamic Scripts Code Editor	140
Figure 6.12: Goal Model Design Tool	141
Figure 6.13: Sales Invoice UI Initial (a) and Simplified (b) Versions	142
Figure 6.14: Sales Invoice UI Simplified for the Sales Officer Role.....	144
Figure 7.1: An Example on Adapting the Product Store UI of OFBiz	150

Figure 7.2: Integrating Adaptive User Interface Capabilities (RBUIs) in OFBiz based on the Cedar Architecture.....	154
Figure 7.3: Saturation Point for Mapping Rules	160
Figure 7.4: Results of the Efficiency Test on Three OFBiz UIs Using Four Example Adaptations	167
Figure 7.5: Box-plot of Load-Testing Results (showing medians)	168
Figure 7.6: Curve of the Load-Testing Results (showing means).....	169
Figure 7.7: UI Adaptation Process: Design-Time versus Runtime UI Adaptation Cycles (based on interviewing an industry expert).....	173
Figure 7.8: Mean Number of Days for One UI Adaptation Cycle from Three Real-Life Enterprise Application Projects Running in Parallel	174
Figure 7.9: Customer Maintenance UI Initial (a) and Simplified (b) Versions	178
Figure 7.10: Participant Demographic Information for Online Usability Study	179
Figure 7.11: Online Usability Study Participant Instructions for the Initial (Left) and Simplified (Right) User Interface Versions.....	180
Figure 7.12: End-User Satisfaction and Efficiency Results.....	181
Figure 7.13: Material UI Initial (a) and Simplified (b) Versions.....	185
Figure 7.14: Vendor UI Initial (a) and Simplified (b) Versions	186
Figure 7.15: Sales Transaction UI Initial (a) and Simplified (b) Versions.....	187
Figure 7.16: Participant Demographic Information for Lab-Based Usability Study	188
Figure 7.17: Lab-Based Usability Study Participant Instructions of the Feature-Set Minimization Part for the Material UI (Left) and Vendor UI (Right)	189
Figure 7.18: End-User Satisfaction Results for Lab-Based Usability Study ...	191
Figure 7.19: End-User Efficiency Results for Lab-Based Usability	192
Figure 7.20: Aggregated Product Reaction Card Results for Lab-Based Usability Study	193

Figure 7.21: Product Reaction Cards Selected More than Two Times by Participants	194
Figure 7.22: Eye-Tracking Results of Fixation Duration and Fixation Count	197
Figure 7.23: Heat Maps Showing an Aggregation of the Participants' Gazing	198
Figure 7.24: Gaze Plots of One Participant with Data Close to the Mean.....	199

List of Tables

Table 1.1: UI Related ERP Implementation Critical Success Factors	8
Table 2.1: Comparison between Approaches to User Interface Development ...	28
Table 2.2: Criteria for Evaluating Adaptive Model-Driven UI Development Systems	37
Table 2.3: Visual Evaluation and Comparison of Adaptive Model-Driven User Interface Reference Architectures	45
Table 2.4: Visual Evaluation and Comparison of Adaptive Model-Driven User Interface Layout Optimization Techniques	54
Table 2.5: Visual Evaluation and Comparison of Adaptive Model-Driven User Interface Development Tools	60
Table 5.1: Cost Matrix Example for Multi-Aspect Trade-Off Analysis	115
Table 5.2: Pareto Optimal Sets of Factors and their Costs for the Adaptation Aspects	120
Table 5.3: Three Example Scenarios with Different Weights for the Adaptation Aspects	121
Table 5.4: Pareto Optimal Sets of Factors and their Costs for Three Scenarios	121
Table 7.1: Some of the Characteristics of OFBiz	153
Table 7.2: Example User Interface Adaptation Operations	161
Table 7.3: Integration Time of Different Adaptation Approaches	163
Table 7.4: Results Obtained from Calculating the Change Impact Metric	164
Table 7.5: Backward Compatibility of UI Adaptation Approaches	164
Table 7.6: Improvement in End-User Satisfaction and Efficiency after UI Simplification	182

Table 7.7: Results of Wilcoxon Signed Ranks Test for Satisfaction and Efficiency	190
Table 7.8: Improvement in End-User Satisfaction after UI Simplification	191
Table 7.9: Improvement in End-User Efficiency after UI Simplification	192
Table 7.10: Improvement in End-User Effectiveness after UI Simplification .	195
Table 7.11: Improvement in Fixation Data after Simplification.....	197

List of Equations

Equation 5.1: Cartesian Product in a General Form	115
Equation 5.2: Cartesian Product of Example Adaptation Factors	116
Equation 5.3: Cost as a Function of an Adaptation Aspect and a Set of Factors	116
Equation 5.4: Cost as a Function of a UI and an Element on the Pareto Front	117
Equation 7.1: Pareto Principle for Mapping Rule Detection Metric	159
Equation 7.2: Approximate Mapping Rule Detection Saturation Point Metric	159
Equation 7.3: Lines-of-Code Metric.....	161
Equation 7.4: Change Impact Metric	162
Equation 7.5: Backward Compatibility Metric	165
Equation 7.6: Runtime Efficiency Metric for UI Adaptation.....	166

List of Code Listings

Listing 5.1: Feature-Set Minimization (Excerpt).....	102
Listing 5.2: Task Model Constraint Example using SQL.....	103
Listing 5.3: Layout Optimization (Excerpt)	108
Listing 5.4: Iron Python Script for Changing the Accessibility of Functions...	110
Listing 5.5: Analyze Multi-Aspect Trade-Offs and Adapt UI (Excerpt)	119
Listing 5.6: Algorithm for Refitting the Layout of an Adapted UI (Excerpt)...	124
Listing 7.1: Code for Reverse Engineering HTML UI to a Model-Driven Representation: Excerpt of HTML Table Example.....	155
Listing 7.2: Code for Enabling Adaptive UI Capabilities	156
Listing 7.3: API Code for Applying the Adapted UI: Excerpt of Widget Hiding Example	157

List of Abbreviations

API	Application Programming Interface
AUI	Abstract User Interface
CRM	Customer Relationship Management
CSF	Critical Success Factor
CTT	Concurrent Task Tree
CUI	Concrete User Interface
ERP	Enterprise Resource Planning
FUI	Final User Interface
IDE	Integrated Development Environment
IT	Information Technology
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
RBUIS	Role-Based User Interface Simplification
RIA	Rich Internet Application
UI / GUI	User Interface / Graphical User Interface
UIDL	User Interface Description Language
UIML	User Interface Markup Language
UsiXML	USer Interface eXtensible Markup Language
WIMP	Window Icon Menu Pointer
XSLT	EXtensible Stylesheet Language Transformations

Glossary

The following list includes definitions for key terms, which are frequently used throughout this thesis:

- **Abstract User Interface (AUI) Model:** represents the UI independent of any modality such as: graphical, voice, gesture, etc. The AUI model relates to the Platform Independent Model (PIM) in the Model-Driven Architecture.
- **Adaptive User Interface:** is aware of its context-of-use and is capable of providing an automatic response to the changes in this context (Fonseca 2010). Such responses could range from a simple layout adjustment to a change in the UI's functionality.
- **Adaptable User Interface:** allows interested stakeholders to manually adapt the desired characteristics. A simple example of adaptable behavior is a software application that supports the manual customization of its toolbars by adding and removing buttons.
- **Aspect (Adaptation):** is used in this thesis for referring to context-of-use facets, which can drive UI adaptations. Some examples include: computer literacy, culture, motor-abilities, screen-size, etc. It is not related to Aspect Oriented Programming unless explicitly specified otherwise.
- **Cedar Architecture:** is presented in this thesis as a reference for developing adaptive model-driven enterprise application user interfaces.
- **Cedar Studio:** is presented in this thesis as an integrated development environment that provides tool support for building adaptive model-driven enterprise application UIs using RBUIS based on the Cedar Architecture.
- **Concrete User Interface (CUI) Model:** is a modality dependent representation of the UI. For example, it can represent the UI in terms of graphical widgets such as: buttons, labels, etc. The CUI model relates to the Platform Specific Model (PSM) in the Model-Driven Architecture.

- **Concurrent Task Trees (CTT)**: is a diagrammatic notation for specifying task models (Paternò et al. 1997).
- **Context-of-Use**: in the field of context-aware computing, is composed of the triplet: *user* who is using the system, *platform* on which the system is running, and *environment* surrounding the system (Calvary et al. 2003).
- **Crisp Goal**: is satisfied with a Boolean constraint such as whether or not to make a UI widget visible.
- **Default Policy**: In RBUIS, a fixed role called “All-Roles” is implicitly allocated to all the software applications’ tasks in all the task models. This role grants access by default to all the application’s tasks for all the roles and can be revoked when necessary by explicitly allocating roles to tasks.
- **Enterprise Application**: generally serves various purposes in managing what is known as an enterprise’s functional business areas such as: accounting, finance, human resources, inventory, marketing, etc. (Oz 2008). Common examples of enterprise applications include: enterprise resource planning (ERP) and customer relationship management (CRM) systems.
- **Factor** (Adaptation): is a UI characteristic that can get affected by an adaptation. For example factors can include: accessibility of functions, font-size, information density, widget type, etc.
- **Feature-Set**: is the set of all the features in a user interface. Features are represented as tasks in the task model.
- **Feature-Set Minimization**: is the process of reducing the feature-set of a UI to contain the minimum number of features, which are required in a certain context-of-use.
- **Final User Interface (FUI)**: represents the actual UI rendered with a presentation technology such as: HTML, Windows Forms, WPF, Swing, etc.
- **Fuzzy Goal**: is satisfied by a fuzzy constraint such as choosing whether a selection widget should be represented as a: combo-box, list-box, or radio buttons, based on different costs pertaining to the context-of-use.
- **Goal**: is used in this thesis to represent an objective that should be achieved by adapting the UI. A goal can be either fuzzy or crisp.

-
- **Integration** (adaptive UI capabilities): is used in this thesis to indicate the empowerment of legacy enterprise applications with adaptive UI capabilities by incorporating our proposed UI adaption technique (RBUIIS) within them.
 - **Layout Optimization**: is the process of producing an optimal UI for a particular context-of-use by adapting the properties of the widgets in the concrete UI model. For example a layout optimization can adapt the: accessibility of functions, font-size, information density, widget type, etc.
 - **Mapping Rules**: are defined for linking the elements between one UI level of abstraction and another. For example, mapping the tasks in a task model to their respective AUI elements, and mapping the AUI elements to their respective CUI elements.
 - **Operation** (Goal Model): is part of the goal model notation. We use it in this thesis for representing UI adaptation factors.
 - **Role**: is used in this thesis as an instance of a role group, which represents an adaptation aspect. A role is not necessarily related to the job, which the employee performs in the enterprise. For example, we can have job title roles such as: accountant, cashier, and manager, and we can also have computer literacy roles such as: novice, intermediate, and expert.
 - **Role Based User Interface Simplification (RBUIIS)**: is presented in this thesis as a mechanism for improving usability through adaptive behavior, by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use.
 - **Role Group**: is used in this thesis as a technical representation of an adaptation aspect, and can contain multiple roles. For example, a computer literacy role group can contain the roles: novice, intermediate, and expert.
 - **Task**: is a UI activity, which is represented as part of a task model. The ConcurTaskTrees notation has the following task categories¹: (1) user task “*is an internal cognitive activity, such as selecting a strategy to solve a problem*”, (2) system task “*is performed by the application itself, such as generating the*
-

¹ Model-Based User Interface – Task Models: w3.org/TR/task-models

results of a query”, (3), interaction task “is a user action that may result in immediate system feedback, such as editing a diagram”, (4) abstract task “has subtasks belonging to different categories, and thus cannot be allocated uniquely using the previous three categories”.

- **Task Model:** is the highest level of abstraction that represents UI features as tasks. A possible representation for task models is the ConcurTaskTrees (Paterno` 1999) notation. This level of abstraction relates to the Computation Independent Model (CIM) in the Model-Driven Architecture.
- **Workflow** (Adaptive Behavior): is used in this thesis for representing adaptive UI behavior through programming constructs such as: control structures, error handling, etc. We apply workflows to the CUI model in order to adapt the widgets’ properties. A simple example of a workflow could be one that contains: (1) a “for” loop that iterates on the CUI widgets, (2) an “if” statement nested in the “for” loop for checking whether a widget is a combo-box, and (3) an assignment nested in the “if” statement to change the type of combo-box widgets to “list-box”.

1

Introduction: Enterprise Application Usability Problems and Solutions

“The beginning is the most important part of the work.”

— Plato, The Republic

Modern businesses greatly depend on enterprise applications such as: enterprise resource planning (ERP) and customer relationship management (CRM) systems, for managing their daily business activities. This dependence drives business owners to demand high quality software products, which allow their employees (end-users) to work without being inhibited by usability or other software-related problems. One would expect this demand is met considering that enterprise applications form an industry with yearly revenues in billions of United States (US) dollars. However, these applications are plagued by usability problems that negatively impact end-user satisfaction and hinder the efficient fulfillment of users' daily tasks. One of the main causes behind this problem is that enterprise applications contain many complex off-the-shelf user interfaces (UIs), while there is a variation in the UI requirements among the different enterprises and even end-users from the same enterprise. A single UI design might be incapable of accommodating such variability. Yet, enterprise applications can contain thousands of UIs making it challenging for software companies to manually develop and maintain all the variations required for improving the usability without increasing the development costs. Furthermore, the scope of UI variability might not be known at design-time.

Adaptive user interfaces have been presented by many research works as a solution for addressing some usability problems by adapting the UIs of software systems to the context-of-use (user, platform, and environment). Adaptive UIs

could offer an effective solution for addressing some context-related usability problems in large-scale software systems such as enterprise applications.

This chapter starts by providing definitions for a few general terms that are frequently used in this work. Afterwards, the research problem, motivation, and objectives are explained. Finally, the thesis organization is given.

1.1 Definitions: Enterprise Applications and Adaptive UIs

We start by defining the terms enterprise, enterprise application, and adaptive user interface (UI) since they are the target of our research.

1.1.1 What is an Enterprise?

Leon (2008) defines an **enterprise** as: “*a group of **people** with a **common goal**, which has certain **resources** at its disposal to achieve this goal*” (p. 7).

By following this definition in our research, the term **enterprise** could refer to small and medium-sized businesses such as retail stores or large corporations such as multi-national financial institutions.

1.1.2 What are Enterprise Applications?

Enterprise applications generally serve various purposes in managing what is known as an enterprise’s functional business areas such as: accounting, finance, human resources, inventory, marketing, etc. (Oz 2008)

Enterprise Resource Planning (ERP) systems are a common example of enterprise applications. Leon (2008) defines an ERP as: “*a set of tools and processes that integrates departments and functions across a company into a common computer system*” (p. 29). These systems can be very large-scale by embodying millions of lines-of-code and thousands of user interfaces.

1.1.3 The Characteristics of Enterprise Application User Interfaces

Enterprise applications generally have box-like WIMP user interfaces, which are mostly used for managing enterprise data. Some examples can be UIs for managing: customer information, bank account information, sales invoices, etc.

The contributions made in this thesis can work for software systems other than enterprise applications given that the same box-like WIMP UI paradigm is used. However, we specifically target enterprise applications considering the large number of complex UIs, which are present in these systems.

1.1.4 What are Adaptive User Interfaces?

Cheng et al. (2009) consider that as a consequence of the evolution towards ultra-large-scale systems, “*software systems must become more versatile, flexible, resilient, dependable, robust, energy-efficient, recoverable, customizable, configurable, and self-optimizing by adapting to changing operational contexts, environments or system characteristics*” (p. 1).

The **context-of-use**, in the field of context-aware computing, is composed of the triplet: user, platform, and environment (Calvary et al. 2003).

Adaptive user interfaces are aware of their context-of-use and are capable of providing an automatic response to the changes in this context (Fonseca 2010). Such responses could range from a simple layout adjustment to a change in the UI’s functionality. After observing the existing literature we were able to differentiate between the following types of UI adaptation solutions:

- **Adaptable user interfaces** allow interested stakeholders to manually adapt the desired characteristics. A simple example of adaptable behavior is a software application that supports the manual customization of its toolbars by adding and removing buttons.
- **Semi-automated adaptive user interfaces** automatically react to a change in the context-of-use by changing one or more of their characteristics using a predefined set of adaptation rules. For example, an application can use a sensor to measure the distance between the end-user and a display

device, then trigger predefined adaptation rules to adjust the font-size accordingly.

- **Fully-automated adaptive user interfaces** can also automatically react to a change in the context-of-use. However, the adaptation would employ a learning mechanism, which makes use of data that is logged over time. One simple example could be a software application, which logs the number of times each end-user clicks on its toolbar buttons and automatically reorders these buttons differently for each end-user according to the usage frequency.

When used in this thesis, the term adaptive user interfaces refers to both the semi-automated and fully-automated solutions.

1.2 Problem Definition

Among the various components of software applications, the user interface is especially important since it connects the end-users to the functionality. A software application's usability is defined in terms of its end-users' effectiveness, efficiency, and satisfaction (ISO 9241 2008). Many software applications could be well-tailored and robust but would eventually fail due to usability problems.

Enterprise applications are plagued by many usability problems (Topi et al. 2005). Some of these problems are due to their complex UIs, which are used in different contexts-of-use. These applications are feature-bloated and are sold as generic off-the-shelf products to be used by people whose diverse needs in the required feature-set and layout preferences are affected by multiple aspects² such as: computer literacy, culture, motor-abilities, screen-size, etc. For example, different users could require a variable part of the software's feature-set. Therefore, displaying a significant subset of the UI could help the end-users in fulfilling their daily tasks more efficiently. Another example is the case where some novice users prefer the UI to be displayed as a step-by-step wizard,

² The word "aspect" is used in this thesis for referring to context-of-use facets that can drive UI adaptations. Some examples include: computer literacy, culture, motor-abilities, screen-size, etc. It is not related to the concept of Aspect Oriented Programming unless explicitly specified otherwise.

whereas advanced users feel more productive if all the UI widgets are displayed on one page. However, the scope of variability might not be known at design-time and it could be costly to develop the UI variations manually.

Adaptive UIs have been presented by many research works as a solution for addressing some usability problems by adapting the UIs of software systems to the context-of-use. Enterprise applications can directly benefit from adaptive UIs for improving their usability. Since these applications could encompass thousands of interfaces, adaptive UIs (i.e., semi/fully-automated adaptation) could be a more feasible adaptation approach than adaptable (i.e., manual adaptation) ones.

Furthermore, providing an approach for developing adaptive UIs is not enough when considering mature legacy enterprise applications. The proposed approach has to integrate within existing legacy systems, in order to empower them with adaptive UI capabilities. Additionally, the integration method should work without incurring a high development cost or significantly changing the way the legacy systems function. This integration challenge must be overcome to allow legacy enterprise applications to benefit from adaptive UIs at a reasonable cost.

1.3 Research Motivation

This section discusses the motivation behind the research based on the adoption rate and the revenues of enterprise applications, and the usability problems that plague these systems.

1.3.1 Enterprise Application Revenues and Adoption Rate

One motivation for researching enterprise applications in general is their wide adoption rate and importance in managing modern businesses, which is reflected by their yearly revenues. For example, the worldwide yearly revenues of enterprise applications are in billions of US dollars, and are expected to increase further in the coming years as shown in Figure 1.1.

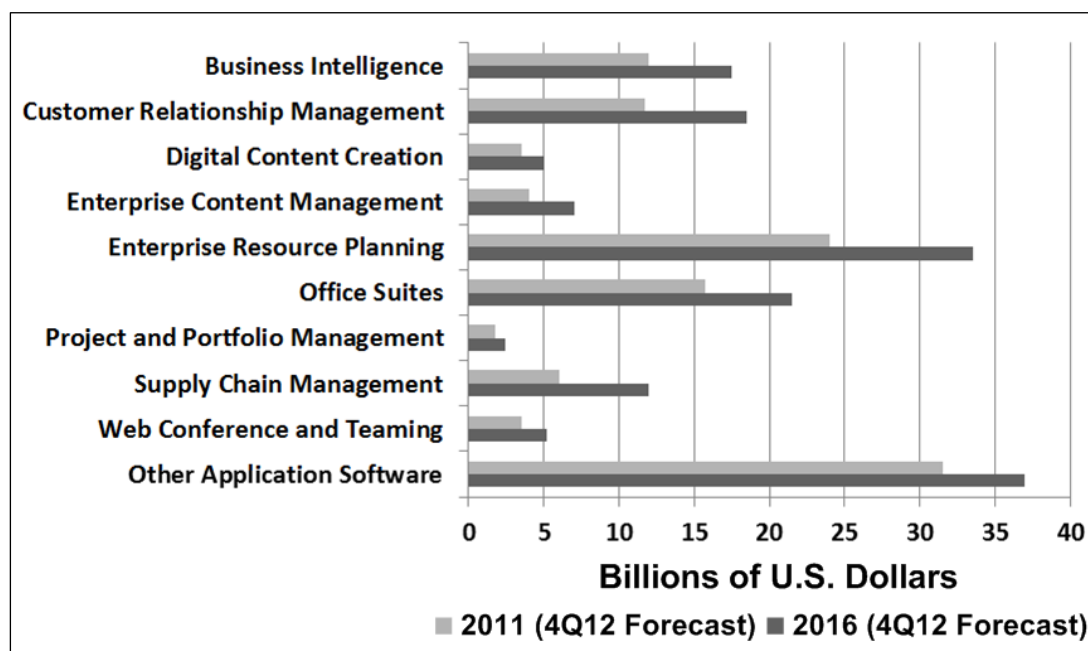


Figure 1.1: Enterprise Application Software Worldwide Revenue Forecast Comparison by Segment between the Years 2011 and 2016 (Gartner 2013)

1.3.2 Usability Problems Reported in the Literature

The existing literature, which includes both academic research and industry reports, clearly indicates that enterprise applications suffer from numerous usability problems.

A survey on enterprise application usability was carried out by the International Data Corporation (IDC) in Denmark, Norway, and Sweden (Lykkegaard & Elbak 2011). It involved 300 enterprise application users in organizations with annual revenues exceeding 100 million US dollars. The results showed that 40% of the participants find enterprise applications difficult to use to a certain extent.

Topi et al. (2005) identified several usability problems that affect the implementation of ERP systems. One of these problems is the overall complexity of ERP UIs, which makes some end-users feel very intimidated when trying to use these systems. Intimidating UIs create usability problems, which could cause enterprise application implementations to fail.

The functionality of software applications tends to increase with every release, thereby increasing the visual complexity (McGrenere et al. 2007). This

phenomenon is referred to as “bloated software” or “bloatware” (McGrenere 2000) and is highly applicable to feature-rich enterprise applications. The presence of UI bloat could make end-users less efficient and frustrated due to the time it takes to locate certain features. Additionally, end-users could become less effective due to being lost when using a bloated UI.

Singh & Wesson (2009) conducted a study to assess the usability of ERP systems and reported several problems mainly related to UI navigation and presentation. One of these problems is the inability of enterprise application UIs to support end-users with different levels of skill (e.g., novices, experts, etc.). The researchers proposed the use of dynamic UI adaptation based on the interactions of each end-user to solve such usability problems.

Commercial enterprise applications such as SAP (Synactive GmbH 2010) attempt to handle UI bloat and variations in UI layout requirements by providing tools, which support the manual development of multiple UI versions. Alternatively, we are aiming towards devising an adaptive UI solution, which applies adaptive behavior to tailor an individual UI design. Considering the large number of UIs contained in enterprise applications, our approach could help in reducing the adaptation cost.

1.3.3 The Impact of Usability Problems on ERP Implementation Success

In previous work, we conducted a study to assess the critical success factors (CSFs) that could impact the success and failure of ERP system implementations (Akiki et al. 2011). The implementation phase deals with the actual system deployment and includes customizing the ERP for business needs, migrating data, training end-users, etc. The factors were divided into four categories: internal people, external people, process, and technology. The study presented the participants with a questionnaire based on 63 CSFs and was conducted in 15 enterprises from various industries namely: retail, banking, manufacturing (various products), healthcare, food and beverage (sales and restaurants), books, insurance, and computer hardware and accessories. We used the collected data to rank the CSFs according to their impact on the ERP implementation success.

The 63 CSFs included several factors, which are interesting for this thesis due to their relation to user interfaces whether directly or indirectly. The *technology acceptance* factor, which ranked as the number one overall factor affecting ERP implementations is correlated with the extent to which UIs support the variable end-user needs (Panorama Consulting Group 2010). We think that software systems could support this variability better with high *flexibility* and less need to go back to the development house to conduct necessary *customization*. Furthermore, by providing more tailored and easy to use UIs, enterprises would require less intensive *training programs*. Such UIs could keep the implementation *costs* realistic, and could allow end-users to be more efficient hence providing enterprises with a better *return on investment*. The *usability* (efficiency, effectiveness, and satisfaction), *familiarity*, and *multilingual* factors, which are directly related to UIs, also contribute towards making enterprise applications better software systems that are easier to use.

The CSFs related to user interfaces are presented in Table 1.1. The CSFs are ranked both within their category and overall. For example, *technology acceptance* is ranked as the primary factor influencing the success of ERP systems. Its global rank is 1 out of 63 factors included in the study, and its category rank is 1 out of 19 factors within the internal people category.

Table 1.1: UI Related ERP Implementation Critical Success Factors

Critical Success Factor	Category	Category Rank	Overall Rank
Technology Acceptance	Internal People	1 / 19	1 / 63
Flexibility	Technology	1 / 16	3 / 63
Customization	Process	1 / 16	5 / 63
Return on Investment	Internal People	2 / 19	7 / 63
Training Programs	Internal People	3 / 19	9 / 63
Usability	Technology	4 / 16	14 / 63
Familiarity	Technology	5 / 16	20 / 63
Realistic Costs	Technology	6 / 16	24 / 63
Multilingual	Technology	8 / 16	27 / 63

We should note that statistical significance cannot be claimed in that study due to the small sample size ($n=15$). However, the results provide us with a general idea on the negative impact that usability problems could have on the success of the complex and expensive implementations of ERP systems. If we combine these results with what is reported in the literature on enterprise application usability problems (e.g., Section 1.3.2), we can say that there is a good incentive for dedicating research effort to solve these problems.

1.3.4 UI Adaptation can Improve Enterprise Application Usability

Several research works, which do not directly target enterprise applications, presented UI adaptation solutions that improve usability. These works considered different UI adaptation aspects and factors. For example, the factors *accessibility of functions*, *information density*, *text versus graphics*, and *navigation structure* were tackled from a cultural perspective (Reinecke & Bernstein 2011). Another research work adapts *UI layout grouping* and the type of *data selection widget* to each end-user's motor abilities (Gajos et al. 2010). In order to investigate the effect of applying similar adaptations on enterprise application UIs, we conducted a preliminary investigation study using an online interactive survey.

We selected computer literacy as one example adaptation aspect, which could affect the end-users' UI preferences. We compiled a list of factors based on which the UI could be adapted to different levels of computer literacy. The list was formed from factors that were mentioned in the literature and others that we considered relevant for enterprise applications. Although this list is not comprehensive, it allows us to compare different adaptations of the same UI and form an idea about whether such adaptations can improve usability. Since this is a preliminary study, we only tested one component of usability namely end-user satisfaction. We grouped the factors under presentation and navigation, as shown below, due to the impact of these categories on enterprise application usability (Singh & Wesson 2009).

Presentation: **Layout Grouping** (tab-page, sub-window, group-box), **Multi-Record Visualization** (grid, carousel, detailed form), **Simple Selection Widget** (combo, slider, radio-buttons), **Multi-Record Input** (scrolling grid, non-scrolling grid, form), **Accessibility of Functions** (high, medium, low), **Information Density** (high, medium, low), **Text versus Graphics** (text only, image only, image and text)

Navigation: **Multi-Document UI** (new window, new page, new tab), **Searching the UI** (go to widget, filter, filter and re-layout), **Navigation Structure** (menu, tree, panel)

To validate whether there is a variance in satisfaction among end-users with different levels of computer literacy, we devised an online interactive survey to test our factors. The survey had one independent variable namely *computer literacy* with three values: *novice*, *intermediate*, and *expert*. The dependent variables are the previously listed factors with their possible values in addition to an open ended value called “other”, which allows participants to specify any possible value, which was not included in the list.

One limitation of surveys inquiring about different versions of the same user interface is the order in which the participant sees the various versions. Participants generally tend to like the first option that they see hence creating some bias in the survey’s outcome. To avoid this potential bias, we designed our survey to display the different UI options as small randomized snippets all on one page. One example question from the survey is illustrated in Figure 1.2. The options are interactive hence allowing the participant to provide better assessment. Participants were asked to rate each of the options on a seven point Likert scale indicating their satisfaction.

The study was carried-out online. Most of the participants were recruited by promoting the study within The Open University community. Some social networks were also used for promoting the study to other communities.



30%

Please click the stars to rate:

Users with Different Levels of Computer Literacy

We classified participants under three groups novice (n=22), intermediate (n=22), and expert (n=45). A two-way ANOVA was performed to examine the

effect of computer literacy on user interface preferences. There was homogeneity of variance between the groups as assessed by Levene's test for equality of error variances. We report measures that were significant ($p < .05$) and partial eta-squared (η^2) due to its significance in human-computer interaction research (Landauer 1997). Partial eta-squared can be interpreted as a small (.01), medium (.06), or large (.14) effect size (Cohen 1988) (pp. 283 and 355). We highlight the following factors, which showed a statistically significant variance among the three computer literacy participant groups: *multi-document UI* ($F(4,288) = 4.507, p = .002, \eta^2 = .059$), *navigation structure* ($F(4,228) = 4.526, p = .002, \eta^2 = .074$), *UI layout grouping* ($F(4,234) = 3.824, p = .005, \eta^2 = .061$). Upon observing the Quantile-Quantile plots we found the data to be normally distributed with some occasional exceptions. Therefore, we also report the outcome of the Kruskal-Wallis (non-parametric) ANOVA as a confirmation to our results: *multi-document UI* ($H(2) = 14.587, P = 0.01$), *navigation structure* ($H(2) = 8.662, P = 0.013$), *UI layout grouping* ($H(2) = 6.447, P = 0.04$).

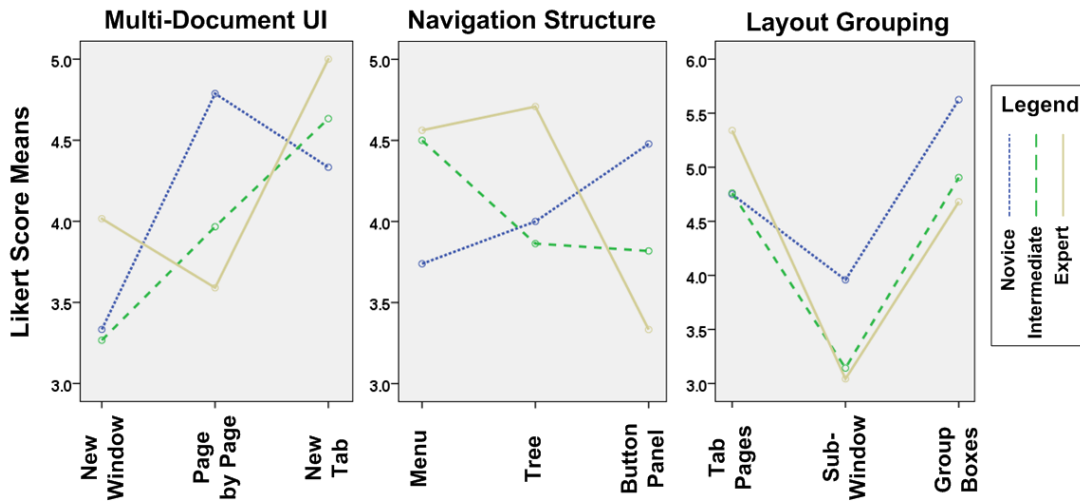


Figure 1.3: Variance in UI Preferences among the Different Computer Literacy Groups

The graphs illustrated in Figure 1.3 show the means of the Likert scores selected by the participants for rating the UI factors, which elicited a statistically significant variance between the different computer literacy groups. The results of this study, in addition to what is reported in the literature hint at the promise of UI adaptation for improving enterprise application usability.

1.4 Research Objectives

This thesis addresses the following overarching research question:

How can adaptive user interfaces be leveraged for improving the usability of enterprise applications?

The **main objective** of this research is to answer the abovementioned research question by devising a general-purpose approach for supporting the development of adaptive enterprise application UIs using runtime models. Therefore, we can say that this thesis contributes a software engineering solution for a human computer interaction (HCI) problem.

The literature review we conducted in Chapter 2 found gaps in the existing state-of-the-art adaptive model-driven UI development systems, which we classified as: reference architectures, adaptation techniques, and support tools. Based on these gaps, the overarching question is divided into sub-questions relevant to the technical characteristics of the contribution and its evaluation. These questions are presented and explained in Chapter 3 alongside the research methodology that is used to answer them.

The adaptive UI development approach contributed by this thesis is divided into the following main parts:

- **Cedar Architecture:** This architecture serves as a reference for stakeholders interested in developing adaptive enterprise application UIs based on interpreted runtime models. We developed a generic service-oriented implementation of this architecture, which could be consumed through an API from different technologies.
- **RBUIS:** The Role-Based User Interface Simplification (RBUIS) mechanism is based on the Cedar Architecture and adapts the UI by minimizing its feature-set and optimizing its layout based on the context-of-use.
- **Cedar Studio:** This tool is an integrated development environment (IDE) for supporting the different stakeholders such as: software developers and IT personnel, interested in developing adaptive model-driven enterprise application UIs using RBUIS and following the Cedar Architecture.

1.5 Thesis Organization

The rest of this thesis is organized into the following chapters:

Chapter 2 — Literature Review: Adaptive Model-Driven User Interface Development Systems: The advantages of the model-driven UI development approach are highlighted. A list of criteria is established and used for evaluating the state-of-the-art adaptive model-driven UI development systems, which we classified into: reference architectures, UI adaptation techniques, and support tools.

Chapter 3 — Research Design: Research Questions, Hypotheses, and Methods: The research questions are established with their hypotheses, and the methods used for answering these questions are stated and justified.

Chapter 4 — The Cedar Architecture: A Reference for Developing Adaptive Model-Driven Enterprise Application User Interfaces: The Cedar Architecture is presented as a reference alongside a general purpose meta-model for supporting the development of adaptive model-driven enterprise application user interfaces.

Chapter 5 — RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior: Role-Based UI Simplification (RBUIS) is presented as mechanism for simplifying enterprise application user interfaces through engineering role-based adaptive behavior. RBUIS is based on the Cedar Architecture.

Chapter 6 — Cedar Studio: An IDE Supporting the Development of Adaptive Model-Driven User Interfaces for Enterprise Applications: The Cedar Studio integrated development environment is described. It supports the development of adaptive model-driven UIs using the RBUIS mechanism.

Chapter 7 — Evaluating the Contributions from the Technical and Human Perspectives: The evaluation of our approach is presented. It covers the technical and human perspectives and uses different research methods.

Chapter 8 – Conclusions and Future Work: The contributions made by this thesis are summarized. Additionally, we provide an overview of possible future work. Preliminary results pertaining to some areas of future work are summarized.

2

Literature Review: Adaptive Model-Driven User Interface Development Systems

“It is what we know already that often prevents us from learning.”

— Claude Bernard

Adaptive user interfaces (UIs) were introduced to address some of the usability problems that plague many software applications. Model-driven engineering formed the basis for most of the systems targeting the development of such UIs. An overview of these systems is presented and a set of criteria is established to evaluate the strengths and shortcomings of the state-of-the-art, which is categorized under architectures, techniques, and tools. A summary of the evaluation is presented in tables that visually illustrate the fulfillment of each criterion by each system. The evaluation identified several gaps in the state-of-the-art and highlighted the areas that can be improved upon.

2.1 Introduction

The user interface (UI) layer is considered one of the key components of software applications since it connects their end-users to the functionality. Well-engineered and robust software applications could eventually fail to be adopted due to a weak UI layer. Some user interface development techniques such as: universal design (Mace et al. 1990), inclusive design (Keates et al. 2000), and design for all (Stephanidis 1997) promote the concept of making one UI design fit as many people as possible. Yet, a UI is dependent on its context-of-use, which is defined in terms of the user, platform, and environment (Calvary et al. 2003). The “one design fits all” approach is unable to accommodate all the cases of variability in the context-of-use, in many cases

leading to a diminished user experience. Building multiple UIs for the same functionality due to context variability is difficult since the scope of variability cannot be completely known at design-time and there is a high cost incurred by manually developing multiple versions of the UI. Adaptive UIs have been promoted as a solution for context variability due to their ability to automatically adapt to the context-of-use at runtime. User interfaces capable of adapting to their context-of-use are also referred to as multi-context or multi-target (Fonseca 2010). A key goal behind adaptive UIs is *plasticity*, denoting a UI's ability to preserve its usability across multiple contexts-of-use (Coutaz 2010). Norcio and Stanley (1989) consider that the idea of an adaptive UI is straightforward since it simply means that: “*The interface should adapt to the user; rather than the user adapting to the system*” (p. 399) but they note that in spite of the simplicity of the definition, there are some difficult and complex problems relating to adaptive UIs. In our study of the literature, we noticed that some of these problems are technical and are related to devising systems that can support the development of adaptive UIs, while others are related to human factors such as the end-user acceptance of these UIs. Realizing the abstract properties illustrated in Figure 2.1 could help in handling some of the technical and human problems related to adaptive UIs.

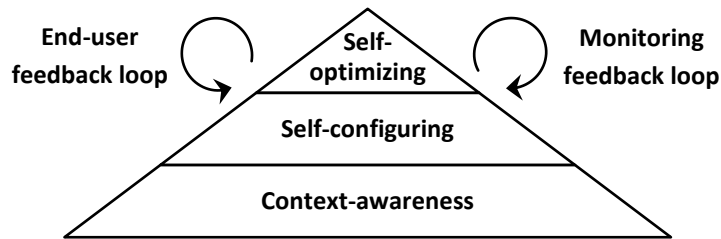


Figure 2.1: Self-* Properties of Adaptive User Interfaces

Salehie and Tahvildari (2009) present a hierarchy of adaptability properties for software systems, referred to as self-* properties. This hierarchy demonstrates different complexity levels in software application adaptability. We consider the following of its properties to be applicable to the domain of adaptive UIs:

- **Context-awareness** “*indicates that a system is aware of its context, which is its operating environment*” (p. 5). If the UI is aware of its context and is able

to detect context changes, then it can trigger adaptations (e.g., based on a set of rules) in response to those changes in order to preserve its usability.

- **Self-configuring** “*is the capability of reconfiguring automatically and dynamically in response to changes*” (p. 5). To keep the UI adaptation rules up to date with an evolving context-of-use (e.g., if a user’s computer skills improve), there is a need for a mechanism that can reconfigure these rules by monitoring such changes. Another type of rule reconfiguration could be based on the end-users’ feedback. For example, the end-user may choose to reverse a UI adaptation or select an alternative. Keeping the end-users involved in the adaptation process could help in increasing their awareness and control, thereby improving their acceptance of the system.
- **Self-optimizing** “*is the capability of managing performance and resource allocation in order to satisfy the requirements of different users*” (p. 5). To adapt this definition to user interfaces, we can say that a UI can self-optimize by adapting some of its properties. For example, adding or removing features, changing layout properties (e.g., size, location, type, etc.), providing new navigation help, etc.

The triplet (user, platform, and environment) forming the context-of-use can be considered as categories of aspects that could promote adaptive UI behavior. The *user* can have an impact on changing the context in terms of variable needs. The needs could be monitored through each user’s behavior upon using the system or be predefined through a set of dynamically configurable rules. For example, the behavior of physically disabled users can be monitored through the speed and accuracy of their mouse clicks and hovering, enabling the UI to be adapted accordingly. On the other hand a user’s countries of birth and residence could be used to adapt the UI according to predefined, dynamically configurable rules based on cultural preferences. The definition of *platform* can accommodate both physical devices (e.g., phone, tablet, laptop, etc.), operating systems, and different types of application platforms (e.g., web, desktop, rich internet application, etc.) (Aquino et al. 2010). Variability in screen size and the available UI widgets are common examples of aspects that could spur platform related adaptive UI behavior. Changes in the *environment* such as: distance

from display devices and mobility, could also incur a change in the context hence requiring the user interface to adapt.

Many applications have usability problems because their UIs do not cater for context variability. Enterprise applications such as enterprise resource planning systems are but one example of such applications (Topi et al. 2005). Adaptive UI behavior has been suggested as a means for enhancing usability in these applications by catering to the variable user needs (Singh & Wesson 2009). Many approaches have been suggested for developing adaptive UIs targeting different types of software systems based on aspects such as: accessibility (Gajos et al. 2010), concurrent tasks (Bihler & Mügge 2007), culture (Reinecke & Bernstein 2011), natural context (Blumendorf et al. 2007), platform (Demeure et al. 2008), etc.

This thesis primarily targets the topic of adaptive UI development systems that adopt a model-driven approach. We mostly focus on the systems that adopt model-driven engineering (MDE) since it offers several advantages and has been receiving the most attention in the literature. Our main aim in this chapter is to demonstrate the strengths and shortcomings of the state-of-the-art.

The scope of this chapter is narrowed down progressively in Sections 2.2 and 2.3. Section 2.2 discusses the different approaches to UI development and adaptation and evaluates these approaches based on criteria from the literature to justify our focus on the model-driven approach. Section 2.3 primarily provides an overview of early model-based UI development systems and justifies why we focused on the latest generation of systems. The evaluation criteria based on which we assess the state-of-the-art is established in Section 2.4 either based on direct recommendations from the literature or by combining features from multiple existing systems.

Our evaluation of the state-of-the-art, which we classified into the dimensions of architectures, techniques, and tools, is presented in Sections 2.5, 2.6, and 2.7 respectively. We believe that a comprehensive system targeting the development of adaptive UIs should provide a reference architecture depicting the various characteristics of the proposed approach, a practical technique to achieve the sought after adaptive behavior based on this reference architecture, and a support tool for stakeholders to develop UIs and adapt them with the proposed

adaptation technique. Finally, a chapter summary is given in Section 2.8 and an overview of our plan for addressing some of the identified limitations is presented.

2.2 Approaches To User Interface Development and adaptation

The existing approaches for developing adaptive user interfaces could be classified under two categories window managers and widget toolkits, and model-driven engineering. Window managers provide a programming model to control the UI's appearance while widget toolkits are reusable code-based libraries of UI components that can support adaptation capabilities. On the other hand, the model-driven engineering (MDE) approach does not rely directly on code for creating the UIs but on higher level specifications from which the UI could be derived. In MDE, the adaptive behavior is usually applied to one of the levels of abstraction before deriving the final UI gets presented to the end-user.

This section provides an overview of the traditional UI development approach and compares the approaches undertaken for developing adaptive UIs. We established the following criteria based on the existing literature and will use them as a basis for evaluating and comparing the approaches:

- **Checking** the adaptive behavior is important to avoid conflicting outputs since this behavior is defined by humans and is thereby error-prone. For example, if a procedure is defined to eliminate part of the UI for a given context-of-use, having the ability to check for a dependency between the removed part and the rest of the UI is important to maintain the UI's functionality (Bergh et al. 2010).
- **Completeness** is defined in terms of the types of UIs that can be produced using a certain UI development approach (Florins 2006). Some approaches might be only suitable for developing a particular type of user interfaces such as WIMP UIs. This criterion could be the same as *generality*, which is the ability of applying the solution to a variety of cases (Myers et al. 2000).
- **Control over the UI** is related to the level of details that the designer can manipulate and the predictability of the final outcome (Florins 2006). Some automated approaches only allow high-level designer input, hence decreasing the control and the predictability of the outcome; while others

allow lower-level input such as control over the concrete widgets. Designer input helps in providing different versions of the UI, one of which is designed by a human and others adapted for a particular purpose. Fully-mechanized UI construction has been criticized in favor of applying the intelligence of human designers for achieving higher usability (Pleuss et al. 2010). It would be better if the designer could manipulate a concrete object rather than its abstraction (Demeure et al. 2009).

- The **cost of developing adaptive UIs** is an important factor that could affect the adoption of this approach. Cost is one of the factors affecting the success of any interactive computer system from the vendor's point of view (Mayhew 1999).
- The **learning curve** is usually affected by how common an approach is in a certain market or software-development company. It has also been related to the *threshold* that indicates how difficult it is to use a certain system for constructing user interfaces (Myers et al. 2000).
- **Technology independence** allows a UI development approach to cover a wider range of existing technologies and to take into consideration new technologies that could emerge in the future. One approach promoting technology independence is UI description languages (UIDLs) such as: UsiXML (Limbourg et al. 2004), UIML (Abrams et al. 1999), etc.
- **Traceability** “is the ability to establish degrees of relationship between two or more products of a development process, especially products having a predecessor-successor or master-subordinate relationship to one another” (Galvao & Goknil 2007) (p. 314). In adaptive UI development, traceability could provide the ability to trace the adaptation performed and be able to revert back to the original user interface either partially or fully.

2.2.1 Traditional Development : Programming, Event, and Markup Languages

Using *programming languages* for user interface development has been investigated for some time. The **Mikey** (Olsen,Jr. 1989) system and its predecessor **MIKE** (Olsen,Jr. 1986) are early propositions for managing user interfaces using programming languages. Mikey provided an example of

applying Pascal to develop UIs for the Apple Macintosh and MIKE was an attempt towards a User Interface Management System (UIMS). Another approach focused on using object-oriented languages (Schmucker 1987). The first attempts in UI development at Xerox PARC used interpreted programming languages such as: Smaltalk and Dlist, which allow developers to easily make changes and test the new UI version. Although this feature was lost with compiled languages like C++, it persists in other languages such as those used for hypertext markup (e.g., HTML).

Event languages allowed developers to control various UI related events (e.g., input and output). Early research work on these languages included the **Sassafras** (Hill 1986) and the **University of Alberta UI Management System** (Green 1985). These types of languages became popular with modern commercial graphical user interface presentation technologies such as: Visual Basic Forms, .NET Windows Forms, Java Swing, etc. Therefore, event languages became part of the visual UI design tools in integrated development environments (IDEs) such as: Visual Studio, Eclipse, etc. Modern languages like the Windows Presentation Foundation (WPF) combine markup languages with programming languages to separate the language used for designing the UI from that used for coding the functionality behind it.

Today, the traditional approach to UI development uses one of the existing presentation technologies. There is a variety of software applications types each relying on different presentation technologies. The following are some examples:

- Desktop Applications: .NET Windows Forms / WPF, Java Swing / AWT, etc.
- Web Applications: HTML, XHTML, CSS, VRML, etc.
- Rich Internet Applications (RIA): Silverlight, XUL, Flex, etc.

Although the traditional development approach has a low learning curve, high completeness, and control over the UI, it has several disadvantages when developing adaptive UIs. The main disadvantages are technological dependency and the high difficulty in adapting the UI to various contexts-of-use without significantly increasing the development cost. Sections 2.2.2 and 2.2.3 present two UI development approaches, namely window managers and widget toolkits, and MDE, which were adopted by many systems for developing adaptive UIs.

2.2.2 Window Managers and Widget Toolkits

Window managers provide developers with a programming model to control the way the UI appears on the screen. However, a direct use of window managers proved to be tedious hence toolkits were developed to make UI construction easier. Toolkits provide a library of widgets and a framework for managing UI creation using this library. Each widget is a component that can manage its own appearance on the screen. Early efforts towards toolkits were the **Apple Macintosh Toolbox** (Huxham et al. 1986) and **Andrew Toolkit** (Palay et al. 1989).

There are approaches that operate on the window level and could be classified as being adaptable rather than adaptive, indicating that manual adaptation is performed by the user. One approach allows HTML-based UIs to be adapted by the end-users through a toolkit with predefined adaptation operations that could store changes in a central repository as Cascading Style Sheets (CSS) (Nebeling & Norrie 2011). UI **Façades** (Stuerzlinger et al. 2006) were presented as a technique for allowing end-users to adapt UIs by dragging and dropping any part of a window to a different location either inside the same window or to another one.

Toolkit-based approaches for adaptive user interfaces have been explored extensively in the literature and attempt to address specific UI adaptation problems. A molecular architecture is offered alongside a toolkit called **Ubit** to provide UI adaptation operations such as: magic lenses, transparent tools, and semantic zooming (Lecolinet 2003). The **caring, sharing widgets** are presented as part of a toolkit that offers widgets with multiple built-in output modalities that can be swapped based on different aspects such as: screen-size, processing power, etc. (Crease et al. 2000). A system called **Fruit** also focuses on multi-modality to support the needs of users with disabilities and those operating in special environments (Kawai et al. 1996). The **selectors** are semantic-based controls that can be presented in a variety of ways in order to replace classical widgets that have a fixed appearance (Johnson 1992). The **ubiquitous interactor** targets device independent UIs by separating the presentation from user interaction and services (Nylander et al. 2004). Widget-level adaptation is also promoted by **WAHID**, which allows the incorporation of adaptive behavior

in new and legacy applications based on internal and external architectures (Jabarin & Graham 2003). Both **ICON** (Dragicevic & Fekete 2001) and **SwingStates** (Appert & Beaudouin-Lafon 2006) are toolkits based on Java Swing. ICON provides an editor that supports the configuration of input devices allowing them to be connected to graphical software interactions, whereas SwingStates uses state-machines to extend existing Java Swing widgets with new interaction techniques.

Widget toolkits reduce the cost of developing adaptive UIs when compared to the traditional development approach and maintain completeness and control. Yet, there are downsides to using this approach. Widget toolkits do not improve technological independence since they are tightly coupled with a single programming language or presentation technology (e.g., selectors with C++, WAHID with MFC, ICON and SwingStates with Java Swing, etc.). Also, they are in many cases hard to extend or non-extensible and do not support traceability. As indicated by Demeure et al. (2008), toolkit-based approaches do not support temporal operators on tasks (e.g., sequence, interleaving, etc.) in a similar manner to MDE (e.g., ConcurTaskTrees (Paterno 1999)), which results in losing the transformation that changes the UI. Another disadvantage of toolkit-based approaches is their inability to perform checking of the overall adaptation impact. One example of such checking is the dependency between UI tasks when the adaptation eliminates certain tasks based on changes in the context-of-use (Bergh et al. 2010).

2.2.3 Model-Driven Engineering

Model-driven development (MDD) is promoted by approaches such as the Model-Driven Architecture (MDA), which provides a technology independent means for absorbing the effect of constant changes in technology and business requirements (Soley & OMG Staff Strategy Group 2000). MDA is about using modeling languages as programming languages rather than merely as design languages since this can improve the productivity, quality, and longevity outlook (Frankel 2003). MDA unites the Object Management Group's (OMG) well-established modeling standards with past, present, and future middleware technologies to integrate "what you have built, with what you are building and

what you are going to build”. Rather than focusing on yet another “next best thing”, MDA raises the bar and designs portability and interoperability into the application at the model level (OMG 2013).

Model-driven engineering (MDE) has a wider scope than MDA’s development activities and combines process and analysis with architectures (Kent 2002). Since MDE aims to raise the level of abstraction of software applications, it can serve as a basis for devising adaptive UIs due to the possibility of applying different types of adaptations on the various levels of abstraction. This approach has received the most attention in the literature. We differentiate between the following model-driven approaches that can be used for developing UIs:

- **Static modeling** relies on models for UI design and eventually ends in a phase before code generation. By definition, static models cannot change at runtime hence are not useful beyond the development phase.
- **Generative runtime modeling** keeps the models alive at runtime to adapt the code-based UI artifacts that were generated at design-time.
- **Interpreted runtime modeling** does not require code generation for creating the UI. Instead, the models are interpreted at runtime to render the UI.

Runtime models constitute an important area of research in model-driven engineering (France & Rumpe 2007). Also, runtime models are usually more suited for supporting adaptive behavior. However, in certain scenarios using runtime models while maintaining the generated code-based artifacts is insufficient for achieving the required adaptations. Some adaptive scenarios require support for actions such as: eliminating widgets; replacing a widget with another; adding new widgets that did not exist during the development phase; or composing a completely new UI from existing UIs. Performing such actions at runtime could be difficult when the user interface is based on generated artifacts. One problem, for example, is the inability to compose new UIs at runtime since the application is expecting to render the UI from code instead of models. Also, substituting a widget with another would be difficult since the types are hard coded, whereas with runtime interpretation the types could be switched in the model and the widget would be rendered accordingly. In contrast, with interpreted runtime models code generation is not needed for creating the UI but the models are interpreted and rendered at runtime thereby

allowing more advanced adaptations. Additionally, by adopting interpreted runtime models, the adaptation could be delegated to a server hence the client machine will be merely responsible for rendering the UI from the adapted model. This method provides a clear separation of concerns. Another benefit of adopting interpreted runtime models is the ability to deploy UI modifications without recompiling the application.

When comparing the MDE approach for UI construction with traditional techniques, Myers et al. (2000) indicate that this approach suffers from a high learning curve. Yet, although the learning curve is generally higher for MDE than traditional development techniques, developers could quickly get used to MDE for devising UIs if the appropriate tool support is provided. Additionally, when assessing whether this learning curve is justifiable we can see that MDE adds value to traditional and toolkit-based approaches by enhancing traceability, technology independence, and the ability to perform checking on the overall outcome of the UI adaptation. We can say that MDE is a viable approach to use and that other research works (e.g., (Florins 2006)) grade it positively in terms of its development costs. Furthermore, Myers et al. (2000) consider that MDE suffers from unpredictability and has a low ceiling indicating that it is incapable of producing advanced UIs. Yet, this consideration is made with fully-automated MDE-based approaches in mind. Nevertheless, other MDE-based approaches apply semi-automated procedures that allow advanced and predictable UIs to be produced by supporting designer input on all levels of abstraction, especially on the concrete UI. Therefore, we consider the ability of MDE to provide good completeness and control over the UI to be dependent on the implementation.

2.2.4 Summary

We can see that each of the previously discussed approaches has some advantages and disadvantages based on the criteria that we established. However, from our analysis of the approaches with respect to the criteria outlined previously (Table 2.1), we think that model-driven engineering is overall better suited for devising adaptive UIs.

Table 2.1: Comparison between Approaches to User Interface Development

	Traditional Development	Widget Toolkits	MDE
Checking			
Completeness			
Control Over the UI			
Development Cost of Adaptive UIs			
Learning Curve			
Technology Independence			
Traceability			

Legend

Good	
Average	
Poor	
Depends on the Implementation	

Due to the advantages provided by the model-driven approach in devising adaptive user interfaces, and due to its wide discussion in the literature we shall dedicate the remainder of the chapter to explore MDE-based adaptive UI development systems. The next section covers early model-based UI development systems in addition to general-purpose frameworks and architectures, which could serve as a basis for modern adaptive model-driven user interface development approaches. In the remainder of the chapter, we explore, evaluate, and compare adaptive UI architectures, techniques and tools that either partially or fully adopt the model-driven approach.

2.3 Background

Many early model-based UI development systems were presented in the literature. We shall briefly discuss their strengths and shortcomings. Additionally, some works have presented frameworks and architectures that can serve as a basis for designing UIs and adaptive systems in general. We also provide an overview of these works and explain the potential of using them for devising adaptive model-driven UIs.

2.3.1 Model-Based User Interface Development

Model-based UI development (MBUID) has been around since the 1980's. Meixner et al. (2011) differentiate between four generations of MBUID systems. The first generation mainly focused on automatically generating UIs but did not provide an integrated MBUID process while the second generation provided developers with the ability to: specify, generate and execute UIs. The third generation mainly focused on the challenge of developing UIs for a variety of interaction platforms and the current (fourth) generation is focusing on the development of context-sensitive UIs. In the current generation of MBUID systems, models and transformations are at the heart of the development process, making UI development model-driven instead of model-based. An existing survey compared and analyzed 14 of the first and second generation MBUID systems, which are mostly concerned with improving model-based UI development or generating UIs from models (Da Silva 2001). Therefore, this sub-section only provides an overview of these early MBUID systems and their strengths and shortcomings while the rest of the chapter tackles recent (4th generation) systems that target the development of adaptive UIs based on a model-driven approach.

2.3.1.1 First and Second Generation MBUID Systems

A number of early systems were presented and primarily focused on improving UI development by making it simpler for developers to devise and maintain user interfaces.

Some systems simply focused on providing better means for UI development. **COUSIN** (Hayes et al. 1985) is a UIMS that targets the development of better quality UIs at a low cost by focusing on providing a level of abstraction in the sequencing of the UI dialog (ordering of input/output events). **ITS** (Wiecha et al. 1990), a four-layer tool-supported architecture, was an early attempt to represent UIs using multiple layers, primarily focused on separating the UI's implementation (actions layer), content (dialog layer), presentation (style rule layer), and interaction (style program layer). The ITS system, allowed the same UI to be presented with multiple styles.

Enhancing the means by which we develop UIs is still an important problem. Yet, the rapid change in the way UIs are developed made such early UIMSs fall victim to the moving targets problem presented by Myers et al. (2000) to indicate that rapid development of technology can make it difficult for tools to keep up the pace.

Another group of systems mainly focused on leveraging MBUID for UI generation. **GUIDE** (Foley et al. 1991) and **HUMANOID** (Szekely et al. 1992) focus on automatic UI generation for allowing designers to experiment with different design possibilities before producing the final user interface. **TADEUS** (Elwert & Schlunbaum 1995) provides a methodology with a supporting environment for generating graphical UIs from a model representing the interactive system. **GENIUS** (Janssen et al. 1993) presented a tool supported technique for generating UIs from data models (entity relationship diagrams) and used a model called dialogue net (based on petri nets) as a visual representation of the UI's dynamics. Other systems supporting UI generation include **JANUS** (Balzert et al. 1996) and **FUSE** (Lonczewski & Schreiber 1996). JANUS also supported the generation of the code that links the UI to the data.

Most of the early MBUID systems targeting automatic UI generation adopted a simple rule-based approach. One exception was **TRIDENT**, which presented tools for automatically generating interactive business application UIs (Vanderdonckt & Bodart 1993) and a generic architecture model for such applications (Bodart et al. 1995). It considered more information for UI generation such as ergonomic rules that are represented using a complex hierarchy. Although such rules provide a more sophisticated generation technique, they could be tedious to implement considering their possibly large number (e.g., 3700 rules (Vanderdonckt & Bodart 1996)).

Some systems worked on improving model-based UI representation. **ADEPT** (Markopoulos et al. 1992) is a design environment that aims to incorporate the theory of modeling (Jacob 1986) instead of just creating a fast prototyping tool. **MASTERMIND** (Szekely et al. 1995) is a UI development environment complementing **HUMANOID** and **GUIDE** and focuses on the presentation model. **MECANO** (Puerta 1996) introduced an interface model called MIM and its modeling language MIMIC.

A common shortcoming in early systems (e.g., COUSIN, GENIUS, HUMANOID, GUIDE, etc.) is the lack of a high level description of the UI, which was represented in different ways such as: application code (HUMANOID), ER diagrams (GENIUS), etc. Such descriptions were later provided by the second wave of systems such as: ADEPT, MASTERMIND, etc. Yet, it was only at the end of the second generation of MBUID systems that task models were introduced to represent UIs at the highest level of abstraction with notations such as the **ConcurTaskTrees** (CTT) (Paternò et al. 1997). Other systems such as **MOBI-D** (Puerta & Eisenstein 1998) investigated new techniques for mapping the task models to lower level UI models. Also, at this stage technology independent languages such as the User Interface Markup Language (**UIML**) (Abrams et al. 1999) were introduced for defining technology independent UI specifications from which technology specific UIs could be generated.

Developing multi-target UIs was considered at a basic level by this generation of MBUID systems. AME (Märtin 1996) offered tool support for the development of interactive systems by constructing UIs from object-oriented analysis models and adapting them to user-specific requirements. AMULET (Myers et al. 1997) is a framework aiming at making multi-operating system UI development easier.

Some earlier systems such as ITS indicated the possibility of adapting UIs for different uses such as: display size, resolution, and color-depth. Yet, UI consistency among different applications is more emphasized than adaptation (e.g., GENIUS, COUSIN, etc.). Even though some later systems support UI adaptation to users and environments (e.g., AME using standardized object classes), this support is more oriented towards manual development rather than adaptive behavior. Therefore, we can say that the major shortcoming in the first and second generations of MBUID systems is that they merely use the model-based approach for UI construction rather than take it further for devising adaptive behavior to support multi-context UIs.

2.3.1.2 Third Generation MBUID Systems

Some domain specific solutions were introduced in this generation such as **Teallach** (Griffiths et al. 2001) that applies the MBUID approach to devise UIs

for object databases. However, the major contribution of this generation was a reference framework that provides guidance for model-driven UI development using multiple levels of abstraction, in addition to the introduction of new UI description languages (UIDLs). Our review does not discuss UIDLs in detail because this is not a contribution of the thesis. A detailed survey of UIDLs can be found in the work of Guerrero-Garcia et al. (2009).

CAMELEON (Calvary et al. 2003) is a unified UI reference framework that is based on two principles (Fonseca 2010): a model-based approach, and the coverage of both the design and runtime phases of multi-target UIs.

CAMELEON is a seminal research work in this generation of MBUID systems. It provides abstraction guidance for devising UIs based on a model-driven approach. As opposed to conventional UI development techniques that merely construct a concrete level (e.g., buttons, text-boxes, etc.), MDE introduces additional levels of abstraction that help in building multi-context UIs.

UIs are represented in CAMELEON on the following levels of abstraction:

- **Tasks and Domain Models:** The task model is the highest level of abstraction that represents UI features as tasks. One possible representation for task models is the **ConcurTaskTrees** (Paterno' 1999) notation that allows tasks to be connected with temporal operators. The domain model denotes the application's universe of discourse and can be represented using UML class diagrams. This level of abstraction relates to the *Computation Independent Model* (CIM) in MDA.
- **Abstract User Interface (AUI) Model:** This level represents the UI independent of any modality such as: graphical, voice, gesture, etc. The AUI model can be represented using UIDLs such as: **TERESA XML** (Berti et al. 2004), **UsiXML** (Limbourg et al. 2004) and **MARIA** (Paterno' et al. 2009) (4th generation). The AUI relates to the *Platform Independent Model* (PIM) in MDA.
- **Concrete User Interface (CUI) Model:** This level is modality dependent. For example, it can represent the UI in terms of graphical widgets such as: buttons, labels, etc. Possible UIDLs for representing concrete user interfaces include: **TERESA XML**, **UIML** (Abrams et al. 1999), **XIML** (Puerta &

Eisenstein 2002), **UsiXML**, and **MARIA**. The CUI relates to MDA's *Platform Specific Model* (PSM).

- **Final User Interface (FUI)**: Represents the actual UI rendered with a presentation technology such as: HTML, Windows Forms, WPF, Swing, etc.

CAMELEON is a suitable reference for approaches that adopt model-driven engineering of interactive systems. MDE can provide a basis for devising adaptive UIs since the levels of abstraction presented by CAMELEON allow different types of adaptive behavior to be implemented such as: Using the task model to adapt the feature-set and using the concrete UI model to adapt the layout.

2.3.2 Reference Architectures for Adaptive Systems

Some software architectures concerned with adaptive system layering can be related to any part (not just the UI) of an adaptive software system. They form a reference for autonomic (self-managing) software systems. We only give a brief overview of these architectures since more details can be found in an existing survey of autonomic software systems (Huebscher & McCann 2008).

The **MAPE-K** loop was created by IBM as a reference model for autonomic computing (IBM 2006). MAPE-K considers software systems as a set of managed resources that can range from an individual application to a more complex cluster. The MAPE-K loop is composed out of four functions with *knowledge* sharing:

- *Monitor*: In this phase, information is collected from the managed resources.
- *Analyze*: Analysis is performed to predict future errors in the system.
- *Plan*: The planning phase prepares the actions required for fulfilling a goal.
- *Execute*: The plan is executed and the dynamic updates are applied.

Rainbow is a framework that employs a control loop for managing self-adaptive systems and provides components that fulfill the phases of the MAPE-K loop (Garlan et al. 2004). Rainbow's architecture layer is made out of the following components:

- The *Model Manager* provides access to the system's architectural model.

- The *Constraint Evaluator* constantly checks the model to see if a constraint has been violated in order to trigger an adaptation.
- The *Adaptation Engine* is responsible for executing the adaptation.

The **Three Layer Architecture** (Kramer & Magee 2007) is an architectural approach and a conceptual reference for self-managing software systems. It comprises the following layers:

- *Component Control* (bottom layer) is a self-managed set of interconnected components capable of reporting its status to the higher levels.
- *Change Management* (middle layer) is responsible for executing actions capable of handling new situations.
- *Goal Management* (highest layer) handles time consuming computations that attempt to achieve an outcome relevant to the sought after goal.

Although these architectures do not particularly target UIs, when combined with UI abstraction frameworks such as CAMELEON they could form the basis for a comprehensive adaptive user interface architecture that can cover both model-driven UI representation and adaptive behavior.

2.3.3 Architectural Patterns and the Separation of Concerns

The patterns discussed in this section are concerned with how the layers of a software system, including the user interface, communicate with each other.

Several implementation architectural patterns, some more recent than others, are available. A common trait between these patterns is the promotion of reusability, and separation of concerns, and the following common tiers:

- **View** represents the user interface “*Presentation Objects*”
- **Model** represents the data “*Domain Objects*”

The following patterns offer different ways of linking the View and the Model.

The **Model-View-Controller** (MVC) architectural pattern is one of the earliest (Xerox PARC late 1970’s) implementation patterns and the most widely discussed in the literature (Krasner & Pope 1988). It provides the Controller as a means for linking the view and the model. MVC could be configured in

different ways, which change the dependency between its components (Martin Fowler 2006). For example in Passive MVC there are no dependencies between the View and the Model hence the View becomes a slave of the Controller, which will be responsible for updating the model and then reloading the view.

The **Model-View-Presenter** (MVP) was developed at IBM for its primary development environments in the 1990's. In MVP, the View relies on the Presenter in order to pass the model data to it and react to user input.

In the **Model-View-ViewModel** (MVVM), the ViewModel acts as a middle man between the View and the Model allowing the user interface to bind to the Model. MVVM was introduced by Microsoft and is largely based on MVC.

Reference architectures for adaptive UIs provide general guidelines about the components of the system. However, by introducing one of the architectural patterns developers will gain more insight on the practical implementation of the systems that intends to follow the proposed architecture.

Despite such patterns and the attempt to achieve a separation of concerns, providing true decoupling between the different tiers in service-oriented applications could be elusive. In such applications, a basic implementation following a three tiered architecture could include the following layers: Server-side (web-services, business logic layer, data access layer, domain model, and data source), client-side (presentation layer, web-service adapters). The web-service adapters on the client-side would act as a link between the client's presentation layer and the Web-Services on the server. Theoretically, by applying such service-oriented architecture we should be obtaining a separation of concerns in addition to decoupling between the client-side and server-side components. Yet in case the UI requires binding to the domain model, which is the case in data-oriented applications (e.g., enterprise applications), the client-side components have to reference a layer similar to the ViewModel resulting in a dependency between the client-side and server-side components. Although a service-oriented architecture is used, whenever a change occurs on the server-side domain model it has to be reflected on the client-side as well. This mandates an upgrade of the presentation layer mainly in desktop applications using presentation technologies such as: Windows Forms, Java Swing, etc.

The abovementioned issues identify a key point in selecting interpreted runtime models for UI development instead of static or even generative runtime models. In case both the interface and data models were based on interpreted runtime models, the client-side components will have true decoupling from the server-side ones. In case the server-side model changes and the UI requires an upgrade, the changes get reflected at runtime without any recompilation or redeployment. These features are already present in HTML-based web applications, which are rendered at runtime. Yet, by using the model-driven approach a general-purpose technology independent solution can be provided.

2.4 Criteria for Evaluating Adaptive Model-Driven User Interface Development Systems

In order to conduct a sound and objective critical review of the existing systems, we setup the following criteria, drawing on direct recommendations from the literature and also by combining features from multiple existing systems.

The criteria we established are presented in Table 2.2 and each is classified under one of the following five categories: UI development, adaptive behavior development, general development support, engaging stakeholders, and output quality. The existing literature on adaptive model-driven UI development systems is quite diverse but we were able to classify each existing work under one or more of the following categories: architectures, techniques, and tools.

Some of the criteria we established are implementation dependent and can only be used to evaluate practical UI adaptation techniques or tools; whereas others are also suitable for evaluating reference architectures as well. Therefore, Table 2.2 indicates the categories (architecture, technique, and tool) to which each criterion is applicable. Two of the criteria, namely completeness and control over the UI, were established in Section 2.2 and are used again since we considered their measure of capability in MDE to be implementation dependent.

We do not claim that our list of criteria is comprehensive. The literature mentions other criteria such as: path of least resistance (Myers et al. 2000) and power in combination (Olsen, Jr. 2007). However, we needed to limit our list to

criteria that we can uniformly apply across the surveyed works using publicly available information.

Table 2.2: Criteria for Evaluating Adaptive Model-Driven UI Development Systems

	Architectures	Techniques	Tools
User Interface Development			
Completeness		X	X
Control over the UI		X	X
Levels of abstraction	X	X	X
Modeling approach	X	X	
Preserving designer input on the UI		X	X
Adaptive Behavior Development			
+Extensibility			
-Adaptation types, aspects, and factors		X	X
-Adaptive behavior	X	X	X
Direct and indirect adaptation	X	X	
Trade-off analysis	X	X	
Visual and code-based adaptive behavior		X	X
Multiple data sources	X	X	
General Development Support			
Modeling, generation, and synchronization			X
IDE style UI			X
+Reducing solution viscosity			
-Flexibility			X
-Expressive leverage			X
-Expressive match			X
Threshold and ceiling			X
Integration in existing systems	X	X	
Engaging Stakeholders			
Empowering new design participants	X	X	X
User feedback on the adapted UI	X	X	
Output Quality			
Scalability		X	

The criteria are listed below and explained. We included one or more of the following codes after each criterion to indicate its applicability to architectures (AR), techniques (TE) and/or tools (TL). These codes reflect the data shown in Table 2.2.

— **Completeness** (*refer to Section 2.2*) (TE, TL)

— **Control over the UI** (*refer to Section 2.2*) (TE, TL)

— Supporting both **direct and indirect adaptation** could make an approach fit for a wider variety of scenarios. User confusion can be reduced by providing the adapted UI as an alternative version (**indirect adaptation**) while maintaining access to the original UI version (McGrenere et al. 2002). Yet, in some software systems such as ubiquitous applications it may be necessary to adapt the UI while the user is working (**direct adaptation**). One example is MASP, which adapts the UIs of smart home systems based on changes in the environment (Feuerstack et al. 2006). (AR, TE)

— **Extensibility** is considered an important characteristic in any new UI development approach (Demeure et al. 2008). We refined its meaning as follows:

— **Extensibility of adaptation types, aspects, and factors** indicates that a UI adaptation approach is not restricted to a single type such as layout optimization but can include a variety of adaptation types such as: feature reduction, navigation help, etc. The approach should also be able to accommodate multiple adaptation aspects such as: cognition, accessibility, natural context, etc. It is also important to support the adaptation of any UI related factor such as: level of access to functions, level of UI details, layout grouping, widget type, font-size, etc. (TE, TL)

— **Extensibility of adaptive behavior** is the approach's capability to add new adaptive behavior at runtime as needed to support a variety of aspects and factors. Contrary to this criterion some approaches might provide limited non-extensible adaptive behavior that is integrated within the system. (AR, TE, TL)

— **Empowering new design participants** by introducing new populations to the design process (Olsen,Jr. 2007): In the case of adaptive user interfaces, new design participants could be non-developers such as: end-users, IT personnel, etc. For example, leveraging communities through crowdsourcing could prove useful for applications that require a lot of effort for defining the adaptive behavior. (AR, TE, TL)

— An **integrated development environment (IDE) style UI** (e.g., similar to Visual Studio or Eclipse) could provide the necessary ease-of-use for managing the UI and adaptive behavior artifacts of large-scale software systems. Developer familiarity and efficiency could be maintained if the tools supporting adaptive model-driven UI development adopt an interface style similar to that of the commercial IDEs. (TL)

— An approach that can **integrate in existing systems** without incurring a high integration cost or significantly changing the system could have a higher adoption rate since many systems are at a mature stage in their development life-cycle. Providing a new advance while maintaining legacy code is a good thing (Olsen,Jr. 2007). (AR, TE)

— Supporting multiple **levels of abstraction** as suggested by CAMELEON (Calvary et al. 2003) offers independence of the implementation (task model), modality (abstract UI), and technology (concrete UI). Also, different levels may be more suitable for certain types of adaptation. Features can be reduced by adapting the highest level (e.g., product-line engineering (Pleuss et al. 2010)) and the layout can be optimized using various levels (e.g., graceful degradation (Florins & Vanderdonckt 2004)). (AR, TE, TL)

— The selected **modeling approach** is important. Supporting *interpreted runtime modeling* allows more advanced adaptations to be conducted (Section 2.2.3). Additionally, one of the major drawbacks of generative modeling approaches is that, over time, models may get out of sync with the running code (Coutaz 2010). (AR, TE)

— **Modeling, generation, and synchronization** of all the levels of abstraction: Model-driven UI development tools should offer developers easy-to-

use WYSIWYG editors and make transformations transparent to provide a better understanding of their effects (Meixner et al. 2011). Tool-supported automated approaches must provide **predictability** to the developers using it (Myers et al. 2000), which in this case can be related to supporting WYSIWYG editors and transformation transparency. (TL)

— Supporting **multiple data sources** allows adaptations to be carried out in various situations. Adaptive behavior models can embody data based on studies, which is the case of adapting UIs to cultural preferences by MOCCA (Reinecke & Bernstein 2011). Also, monitoring the user’s behavior allows models to evolve and can be beneficial in other situations (e.g., targeting accessibility with MyUI (Peissner et al. 2012)). (AR, TE)

— **Preserving designer input on the UI** is important since automated choices without a rationale make adaptive UIs unpredictable. The success of UI development techniques could be negatively impacted by unpredictability (Myers et al. 2000). UI adaptations will obviously override the input made by the designer. Yet, in some cases designers might want to preserve some characteristics (e.g., prioritizing the size of a widget over others), thereby enhancing the predictability of the outcome. (TE, TL)

— **Reducing solution viscosity** is achieved if a tool reduces the effort required to iterate on the possible solutions based on the following criteria (Olsen,Jr. 2007):

— **Flexibility** denotes the ability to “*make rapid design changes that can then be evaluated by the users*” (p. 255). The tools we are evaluating should be *flexible* by providing the ability to devise both the UI models and the adaptive behavior in a way that allows easy testing and refinement. (TL)

— **Expressive leverage** “*is where the designer can accomplish more by expressing less*” (p. 255). We consider that expressive leverage can be achieved by promoting the reusability of UI model parts (e.g., the same way visual-components are reused in traditional IDEs) and the adaptive behavior (e.g., visual-parts or scripts). (TL)

- **Expressive match** “*is an estimate of how close the means for expressing design choices are related to the problem being solved*” (p. 255). One way to improve expressive match is by supporting visual-design tools for the UI models and innovative means for specifying *adaptive behavior visually*. (TL)
- **Scalability** is an important criterion that must be checked for every new system (Olsen,Jr. 2007). If the scalability of an adaptation technique is not demonstrated using real-life scenarios, its adoption for complex software systems could decrease. (TE)
- An ideal tool would have **low threshold and high ceiling** (Myers et al. 2000). The “threshold” represents the difficulty in learning and using the tool, and the “ceiling” relates to how advanced the tool’s outcome can be. (TL)
- **Trade-off analysis** between several potentially conflicting adaptations is essential for producing an optimal UI, especially in systems that target multiple adaptation aspects and factors. One example described in the literature is the trade-off between the user’s vision and motor abilities (Gajos et al. 2007). (AR, TE)
- **User feedback on the adapted UI** keeps the end-users in the loop of the adaptation process and provides awareness of automated adaptation decisions and the ability to override them when necessary. Keeping humans in the loop is considered one of the principles of adapting UIs based on MDE (Balme et al. 2004). It can increase the end-users’ UI control (McGrenere et al. 2002) and feature-awareness (Findlater & McGrenere 2007) affected by adaptive/reduction mechanisms. (AR, TE)
- **Visual and code-based** representations allow different stakeholders such as: developers and IT personnel, to implement **adaptive behavior**. Some techniques only support a textual representation such as cascading style sheets in Comet(s) (Demeure et al. 2008) and behavior matrices in FAME (Duarte & Carriço 2006). Yet, others indicate that a visual notation can greatly simplify the creation of UI adaptation rules by hiding the complexity of programming languages (López-Jaquero et al. 2009). (TE, TL)

We now use these criteria to evaluate the fourth (current) generation adaptive model-driven UI development systems. The reference architectures, practical techniques, and supporting tools are evaluated in Sections 2.5, 2.6, and 2.7 respectively.

2.5 Reference Architectures for Adaptive User Interfaces

Architectures play a fundamental role in self-adaptive software systems (Oreizy et al. 1999). An architecture-based approach is promoted for these systems (Kramer & Magee 2007) since it could build on existing work and offer generality, abstraction, scalability, etc. Following a reference architecture could help in realizing adaptive UIs in complex systems. Several architectures have been proposed as a reference for applications targeting adaptive UIs and other UI related features such as: multimodality, distribution, etc. This section focuses on evaluating and comparing existing research works related to architectures of adaptive UIs. We briefly describe these architectures and argue their strengths and shortcomings, and conclude with a comparison between them. We should note that this section only evaluates reference architectures. Existing adaptive UI techniques, whether based on a defined architecture or not, are discussed and evaluated in Section 2.6.

2.5.1 Review

The **3-Layer architecture**³ was presented for devising adaptive smart environment user interfaces (Lehmann et al. 2010). Due to the ubiquitous nature of its target applications, this architecture only supports *direct adaptations*. Information is read from sensors, and the environment context pillar is targeted hence *multiple data sources* are not supported. The *modeling approach* of this architecture is based on generative runtime models, which could be less flexible than interpreted runtime models for performing advanced

³ In order to avoid confusion, we should note that this architecture is not related to the Three Layer Architecture for self-managing software systems presented by Kramer & Magee (2007).

adaptations. Additionally, the work does not specify whether the architecture is meant to support all the *levels of abstraction*.

CAMELEON-RT is a reference architecture model for distributed, migratable, and plastic user interfaces within interactive spaces (Balme et al. 2004). This architecture targets all context-of-use pillars (user, platform, and environment), and could be considered general-purpose due to its implementation neutrality. We consider that it provides a good conceptual representation of the *extensibility of adaptive* behavior through the use of open-adaptive components (Oreizy et al. 1999), which allow new adaptive behavior to be added at runtime. Also, both *direct and indirect adaptations* could in theory be implemented using these components. It follows the CAMELEON framework hence all the *levels of abstraction* are supported. This architecture depicts observers that collect data on the system, user, platform, and environment, and feed it to a situation synthesizer thereby supporting *multiple data sources*. CAMELEON-RT serves as a high-level reference but does not provide implementation specifications on *integrating in existing systems*.

FAME is an architecture that targets adaptive multimodal UIs using a set of context models in combination with user inputs (Duarte & Carriço 2006). It only targets modality adaptation hence it is not meant to be a general-purpose reference for adapting other UI characteristics. The adopted approach allows designer input on the CUI hence providing good *control over the UI*. *Adaptive behavior* can be *extended* using adaptive behavior matrices. FAME depicts support for *multiple data-sources* including device changes, environmental changes, and user inputs that feed into related models. A combination of the multiple data sources and the adaptive behavior matrices should be able to support both *direct and indirect adaptations*.

Malai is an architectural model for interactive systems (Blouin & Beaudoux 2010) and forms a basis for a technique that uses aspect-oriented modeling (AOM) for adapting user interfaces (Blouin et al. 2011). The *extensibility of adaptive behavior* is poor since multiple presentations have to be defined at design-time by the developer, to be later switched at runtime. Although Malai supports multiple *levels of abstraction*, the *modeling approach* relies on generating code (e.g., Swing, .NET, etc.) to represent the UI. Also, it does not

describe *multiple sources* for acquiring adaptive behavior data. In theory, both *direct and indirect adaptations* can be supported. In addition to being technology dependent (a Java example is provided), UI adaptation in Malai is not decoupled from the target software systems thereby requiring significant code modification for adaptive behavior to be *integrated in an existing system*.

We noticed that several criteria were not addressed by any of the works reviewed in this section. The architectures did not incorporate any components for *empowering new design participants* and did not offer insights on managing *trade-offs* between possibly conflicting adaptations. Although reference architectures are not expected to provide an implementation for trade-off analysis, depicting trade-off as part of the architecture could provide a conceptual reference for those wishing to implement a technique based on the architecture. Supporting *user feedback on the adapted UI* is neglected. The “3-Layer Architecture” does not support user feedback but refers to another work (Brdiczka et al. 2007) that does not offer an architecture but uses user-feedback for refining initial situation models at runtime in order to improve the reliability of detected situations. Malai allows developers to define feedback that would help users to understand the state of the interactive system but the user cannot provide feedback on the adaptations (e.g., reverse an unwanted adaptation). In spite of the importance of *integration in existing software systems* that are in a mature development stage, the evaluations were conducted by building new prototypes.

2.5.2 Summary of the Review

After arguing the strengths and limitations of existing adaptive UI architectures, we present a comparison in Table 2.3 that illustrates the extent to which each of the architectures fulfills the criteria we established in Section 2.4.

The criteria that were not addressed by the existing architectures are: *User feedback on the adaptive UI*, *trade-off analysis*, *integrating in existing systems*, and *empowering new design participants*. The remaining criteria were addressed to an extent but there is still room for improvement, especially for the *modeling approach*.

Table 2.3: Visual Evaluation and Comparison of Adaptive Model-Driven User Interface Reference Architectures

Legend									
	Direct and indirect adaptation	Empowering new design participants	Extensibility of adaptive behavior	Integrating in existing systems	Levels of abstraction	Modeling approach	Multiple data sources	Trade-off analysis	User feedback on the adapted UI
● Completely fulfills									
◐ Partially fulfills									
○ Does not fulfill									
○ Not specified									
3-Layer Architecture	◐	○	○	○	○	◐	◐	○	○
CAMELEON-RT	●	○	●	○	●	○	●	○	○
FAME	●	○	●	○	◐	○	●	○	○
Malai	●	○	◐	○	●	◐	◐	○	○

2.6 Techniques for Devising Adaptive Model-Driven UIs

Adaptive behavior can target a variety of UI characteristics. In order to provide a boundary for this work, we shall focus on the techniques related to at least one of the two UI adaptation types that are the most targeted in the literature, namely feature-set adaptation and layout optimization. We define a feature as a functionality of a software system and a minimal feature-set as the set with the least features required by a user to perform a job. An optimal layout is the one that maximizes the satisfaction of constraints imposed by a set of adaptation aspects such as: physical impairments, computer literacy, etc. An optimal layout is achieved by adapting the properties of concrete widgets such as: type, grouping, size, location, etc.

2.6.1 Feature-Set Adaptation Techniques

The functionality of software applications tends to increase with every release hence increasing the visual complexity. This phenomenon, referred to as “bloatware” (McGrenere 2000), has a negative impact on usability especially for users who do not require the complete feature-set. It could be helpful to provide

each end-user with a minimal feature-set that reduces unnecessary bloat present in feature-rich UIs. Since the existing solutions that are related to UI bloat mostly focus on design-time adaptation rather than runtime adaptive behavior, we did not evaluate them according to the criteria established in Section 2.4. Instead, we grouped them into categories and provided a general evaluation of their strengths and shortcomings.

2.6.1.1 Review

Several ***theoretical propositions*** were made for reducing a UI's feature-set based on the context-of-use. Providing a multi-layered user interface design is promoted for achieving *universal usability* (Shneiderman 2003). Other researchers propose using *two UI versions*, one fully-featured and another personalized, for taming the bloat in feature-rich applications (McGrenere et al. 2002). An early research work proposes the use of a “*training wheels*” UI that blocks advanced functionality from novice users (Carroll & Carrithers 1984). These works present a sound theoretical basis, useful for providing the users of feature-bloated software applications with a minimal feature-set. Yet, the given examples, a basic text editor (Shneiderman 2003) and the Word 2000 menu (McGrenere et al. 2002), do not match the complexity of large-scale systems such as enterprise applications. Also, these works do not provide or describe a technical implementation.

Approaches from ***software product-line*** (SPL) engineering (Pleuss et al. 2010) are used to tailor software applications and some particularly address tailoring user interfaces. *MANTRA* (Botterweck 2011) adapts UIs to multiple platforms by generating code particular to each platform from an abstract UI model. Although SPLs can be dynamic (Bencomo et al. 2008), the SPL-based approaches for UI adaptation focus on design-time (product-based) adaptation whereas runtime (role-based) adaptive behavior is not addressed.

Several ***commercial software applications*** use role-based tailoring of the UI's feature-set. *Microsoft Dynamics CRM* (Microsoft 2011) and *SAP's GuiXT* (Synactive GmbH 2010) offer such a mechanism, yet it is not generic enough to be used with other applications and it requires developing and maintaining

multiple UI copies manually. An approach that operates at the model level could be more general-purpose.

Some approaches relied on *decomposition* to break the UIs into smaller fragments that fit the context-of-use better. *Graceful degradation* is used as a method for supporting UIs on multiple devices (Florins & Vanderdonckt 2004) and could be used for decomposing/recomposing UIs. An initial UI is constructed for the platform with the least constraints, and then other versions are generated for the platforms with more constraints based on designer annotations. In concept, this method could be used for minimizing a UI's feature-set by decomposing it into smaller fragments. Yet, its main limitation lies in its reliance on designer annotations that would not work when the adaptations are only known at runtime. An interesting approach would be to support runtime annotations combined with automated procedures that can adapt the UI based on each user's behavior. Another approach called *(de)composition* complements some aspects of graceful degradation (Lepreux et al. 2007). It aims towards supporting reusability at a high-level design without the need for applying constant copy/paste operations. Similar to the graceful degradation approach, *(de)composition* could be in concept used for reducing the UI's feature-set. The authors mention the applicability of this approach both at design-time and runtime but no significant demonstration is made towards runtime scenarios since all the examples were restricted to design-time. Decomposing/composing UIs at runtime while maintaining functionality would require work that does not merely adjust the UI's layout but maintains and adapts the functionality behind it.

2.6.1.2 Summary of the Review

The main limitations in approaches attempting to target feature-set adaptation are: lack of a practical implementation mechanism, lack of generality of the solutions, or restriction to design-time adaptation without offering a runtime adaptive solution. Based on these limitations, we can say that more work is needed to provide a general-purpose, model-driven, and tool supported adaptive UI mechanism capable of reducing UI bloat at runtime by adapting the UI's feature-set based on the context-of-use.

2.6.2 Layout Optimization Techniques

Providing an optimal layout based on the context-of-use could improve usability by catering for the diverse end-user needs. For example the usability of SAP, the world's leading ERP system (Jacobson et al. 2007), is mostly affected by navigation and presentation (Singh & Wesson 2009) and its UI does not adapt to each end-user's skills (Uflacker & Busse 2007). Many existing works use different approaches to target the adaptation of the UI layout. In this subsection, we shall provide a brief description of each of these works and argue their strengths and shortcomings using the criteria we established in Section 2.4.

2.6.2.1 Review

The COntext Mouldable widgET (**Comet(s)**) was introduced as a set of widgets that support UI plasticity (Calvary et al. 2005). It provides an architectural style for plastic UIs by combining the toolkit and model-based approaches (Demeure et al. 2008). A "Comet" is capable of self-adapting or being adapted to the context-of-use.

Comet(s) target the adaptation of individual widgets but does not focus on the entire layout. Centralizing the adaptive mechanism could be more *scalable* than defining it in each widget and could make Comets a more interesting solution for adaptive UI functionality. Using a widget toolkit to represent the CUI provides good *control over the UI* and could theoretically be used to develop different types of UIs that can adapt to any context pillar, therefore providing good *completeness*. The *extensibility of the adaptive behavior* is claimed to be supported through style-sheets but the *adaptation types* are not extensible since each Comet can only adjust its own shape, whereas different types of adaptation (e.g., feature-set, navigation, etc.), which might be more related to the overall user interface design, cannot be supported by this architectural-style. One of the goals of Comet(s) is to sustain UI adaptation at any level of abstraction: tasks and concepts, abstract, concrete, and final UI as elicited in model-driven approaches (Calvary et al. 2003). Therefore, the *levels of abstraction* are embodied in what are referred to as the Logical Consistency (LC), Logical Model (LM), Physical Model (PM), and technology primitives. Comet(s) does not

present a means for reading adaptation data from *multiple data sources* as presented by CAMELEON-RT for example. We consider the *modeling approach* to be poor since it is necessary to have a code-based implementation as opposed to the possibility of using interpreted runtime models. Although it is not explicitly described, the use of style-sheets can support both *direct and indirect adaptations*.

DYNAmic MOdel-bAsed user Interface Development (**DynaMo-AID**) is a: design-process and runtime architecture for devising context-aware UIs and is part of the Dygimes UI framework (Coninx et al. 2003). Its runtime architecture includes three major modules namely context monitoring, functional core, and presentation that are linked by a dialog controller (Clerckx et al. 2005). The final UI is rendered from task models after adapting them to the operating environment and device.

DynaMo-AID is limited to WIMP-style UIs and only targets the environment context hence has low *completeness*. The *levels of abstraction* supported by DynaMo-AID are restricted to the task model from which the final UI is generated. The support of interpreted runtime models provides a good *modeling approach* but since designer input is not supported on the CUI the *control over the UI* could be negatively affected. *Adaptive behavior* is extensible but is restricted to one *type of adaptation*, namely, the UI dialog. Due to the pervasive nature of its target applications (e.g., tourist guide mobile application called Imogl for an open air museum (Clerckx et al. 2006)), DynaMo-AID only supports *direct adaptations* and environment sensors as a *data source*. This architecture is particularly criticized for using what is referred to as a “Task Tree Forest” (Blouin et al. 2011). The critics note that since each tree corresponds to the tasks possible in a given context, the combinatorial explosion would affect the approach’s *scalability* when it is applied to complex systems.

Supple supports automatic generation of UIs adapted to each user’s abilities (e.g., motor and vision), devices, tasks, and preferences (Gajos et al. 2010). It relies on a high-level interface specification, device model, and user traces to generate the UI.

In terms of *completeness* Supple’s approach is particularly well suited for box-like UIs due to the existence of a vocabulary of interactions for this UI type.

However, although not tested, its creators indicate that it is not limited to such UI types especially if new vocabularies of interactions could be identified (Gajos et al. 2010). Supple *interprets and renders UI models* at runtime hence making the fulfillment of more advanced adaptations easier. Yet, the adopted technique generates the UI from a high-level model (one *level of abstraction*), which prevents designer input from being made especially at the CUI level hence provides less *control over the UI*. The inability to have human input at the different levels of abstraction, at least at design-time, makes the approach difficult to adopt for large-scale systems such as enterprise applications. Supple has built-in algorithms for adapting the UI and does not provide a means for *extending the adaptive behavior* through either a *visual or code-based* representation. The only *adaptation type* supported by Supple is layout optimization. Vision and motor capabilities are the primary supported *adaptation aspects*, and 40 UI factors (e.g., font size, widget size, etc.) are supported. Supple does not provide a means for *extending adaptation types, aspects, and factors*. Also, it has been criticized (Peissner et al. 2012) for exceeding acceptable performance times. This criticism could be justified by observing some of its worst-case scenarios that could span over 30 seconds when computing the most appropriate UI layout. This timing is not appropriate for software systems looking for high efficiency. One advantage that Supple has over other systems lies in performing true layout optimization due to its ability to quantify UI quality. The quantification is achieved by using a cost function to compare UI versions in order to determine the most optimal one. This approach also allows Supple to support *trade-off analysis*, which was demonstrated for a fixed number of adaptation aspects, namely motor and vision capabilities (Gajos et al. 2007). Supple is complemented by a system called **Arnauld** (Gajos & Weld 2005), which is responsible for eliciting user preferences in order to adapt the UI at runtime. This process could serve as a *feedback mechanism* but the sole reliance on runtime elicitation can be time consuming and might not provide sufficient data in comparison to leveraging *multiple data sources*. Supple primarily targets *indirect adaptation* since it builds up a user-model over time based on preference elicitation.

The Multi-Access Service Platform (**MASP**) is a UI management system targeting ubiquitous UIs for smart homes (Feuerstack, et al. 2006). MASP uses

a model-driven approach to support: multimodality (Blumendorf et al. 2008), distribution (Blumendorf et al. 2007), synchronization (Blumendorf et al. 2006), and adaptation (Schwartz et al. 2009). Although it demonstrates powerful capabilities in UI distribution and multimodality, we focus on its adaptation capabilities to stay within our scope.

Adopting a box-based layout (Feuerstack et al. 2008) for repositioning different UI segments at runtime using content scaling prevents widget level feature-adaptation and decreases *completeness*. The *modeling approach* bases the final UI on generated code or markup (apache velocity templates) (Blumendorf et al. 2008) hence allowing less advanced adaptations to be performed at runtime as opposed to a fully-dynamic approach. MASP uses *direct adaptation* whenever a context change is detected due to the ubiquitous nature of the target smart home systems. The primary *adaptation type* supported by MASP is layouting based on the several environment-related *aspects* such as distance and spots (e.g., distance from particular physical objects). Also, a limited number of UI adaptation factors are supported (e.g., orientation, size, rearrangement of predefined UI groups, etc.) and no means is provided *for extending the adaptation types, aspects, and factors*. As described in its UI construction technique (Feuerstack 2008), MASP supports all the *levels of abstraction* suggested by CAMELEON. It also supports designer input on the CUI to provide *control over the UI*. Since it targets ubiquitous applications, MASP's *data sources* are restricted to environment sensors as indicated by the 3-Layer architecture (Lehmann et al. 2010). MASP provides a tool to visually divide the layout into boxes but does not support the definition of *visual and code-based adaptation rules*, which could cover a variety of layout optimization factors that go beyond changing the font-size, and layout grouping.

One technique uses aspect-oriented modeling (**AOM**) for adapting UIs (Blouin et al. 2011) based on the **Malai** architecture (reviewed in Section 2.5).

This approach requires several UI presentations to be defined at design-time and a weaver is used to associate these presentations to instrument classes that handle the way the UI functions at runtime. It provides *completeness* because it targets post-WIMP UIs and could logically target others as well since the UI is generated to code, hence also providing good *control over the UI*. The *adaptive*

behavior could be *extended* but this can only be done at design-time since the *modeling approach* relies on code. Hence, the UI variations have to be manually defined by the developer. *Scalability* is demonstrated by taming the combinatorial explosion of complex interactive system adaptations. The meta-model does not support the addition of *adaptation types, aspects, and factors*. Also, no mechanism is provided for adding *adaptive behavior visually*.

MyUI is a user interface development infrastructure for improving accessibility through adaptive UIs (Peissner et al. 2012). It uses an open pattern repository for defining adaptation rules. User interfaces are specified as an abstract model that is represented using a notation based on state charts.

MyUI is presented as a *general-purpose* infrastructure but it was only demonstrated with basic interactive television UIs. It does not support all the *levels of abstraction* suggested by CAMELEON but only relies on an abstract model to automatically generate the final UI. Hence, the designer's *control over the UI* is reduced due to the lack of designer input on the concrete UI. MyUI supports both *direct and indirect adaptations* since the users can swipe a card to let the system identify who they are and customize the UI based on their profile; sensors are also able to detect gestures (e.g., leaning towards the screen due to poor vision) and perform direct UI adaptations accordingly. It is possible to *extend the adaptive behavior* by modifying the state-chart models; however this extension is performed at development-time and could require a redeployment of the application. MyUI shows the possibility of supporting *multiple adaptation data sources* since the patterns it uses for defining the adaptation rules can embody expert knowledge and the system has the ability to acquire environment data using sensors. Although MyUI allows the end-users to reverse the adaptations, its *feedback* mechanism can be enhanced further by offering users an explanation of the reason behind the adaptation. The *adaptive behavior* (adaptation rules) in MyUI are *defined visually* using a state-chart model; however the basic accessibility adaptation examples (e.g., changing font-size) that were presented do not demonstrate whether the state-chart notation has the potential for defining more advanced usability related adaptations such as: changing the layout grouping, widget types, etc.

We noticed that several criteria were not addressed by any of the works reviewed in this section. None of the works suggest a mechanism for *preserving designer input on the UI* after the adaptive behavior has been applied. Also, no ideas are presented for *empowering new design participants* such as engaging and leveraging end-user communities to participate in UI adaptation through the means of crowdsourcing. Aside from Supple, most techniques do not offer any insight on managing *trade-offs* between possibly conflicting adaptations. Supporting *user feedback on the adapted UI* is also neglected. For example, Comet(s) should in concept allow end-users to explore possible design alternatives but this point was left for future work. The techniques we reviewed were evaluated by developing new prototype systems instead of showing the possibility of *integrating in existing software systems*. For example, MASP was evaluated by (re)building home automation applications such as: energy, cooking, and health assistants; Supple was evaluated by developing a variety of simple UI dialogs such as: email client, ribbon and print dialogs, etc. The existing techniques did not offer any mechanisms for *extending adaptation types, aspect, and factors* but merely supported a limited number of them. For example, Supple supports 40 factors and targets a limited number of adaptation aspects related to physical impairments.

2.6.2.2 Summary of the Review

We present a comparison of the layout optimization techniques in Table 2.4. The criteria that were not supported by the state-of-the-art are: *preserving designer input on the concrete UI*, *integrating in existing systems*, *empowering new design participants*, and *extensibility of adaptation types, aspects, and factors*. Also, significant improvement could be made on *trade-off analysis*, *user feedback*, and supporting *visual and code-based adaptive behavior representations*. We noticed that very few works conducted *scalability* tests, which are important to demonstrate if the technique works with large-scale and complex UIs. The remaining criteria were addressed to different extents and some have room for improvement.

Table 2.4: Visual Evaluation and Comparison of Adaptive Model-Driven User Interface Layout Optimization Techniques

	Legend														
	<p>● Completely fulfills</p> <p>◐ Partially fulfills</p> <p>○ Does not fulfill</p> <p>○ Not specified</p>														
	Completeness	Control over the UI	Direct and indirect adaptation	Extensibility of adaptive behavior	Extensibility of adaptation types, aspect, factors	Empowering new design participants	Integrating in existing systems	Levels of abstraction	Modeling approach	Multiple data sources	Preserving designer input	Scalability	Trade-off analysis	User feedback on the adapted UI	Visual and code-based adaptive behavior
Supple	◐	◐	●	○	○	○	○	◐	●	◐	○	◐	◐	●	○
MASP	◐	●	◐	●	○	○	○	●	◐	◐	○	○	○	○	◐
Comet(s)	●	●	●	●	○	○	○	●	◐	◐	○	○	○	○	◐
DynaMo-AID	◐	◐	◐	●	○	○	○	◐	●	◐	○	◐	○	○	◐
AOM (Malai)	●	●	●	◐	○	○	○	●	◐	◐	○	●	○	○	◐
MyUI	◐	◐	●	◐	○	○	○	◐	◐	●	○	○	○	◐	◐

2.7 Tools Supporting Adaptive Model-Driven UI Development

The adoption of a technique depends largely on giving researchers and practitioners the means of applying ideas without resorting to low level implementation (Cheng et al. 2009). The model-driven approach to UI development can serve as a basis for devising adaptive UIs due to the possibility of applying different types of adaptations onto various levels of abstraction. Yet, implementing adaptive model-driven UIs requires the tools that support a definition of the necessary UI models and adaptive behavior. In this regard, existing tools still lack many features required for supporting adaptive model-driven UIs. This section provides an overview of the state-of-the-art tools for developing (adaptive) model-driven UIs and evaluates them according to their support of the criteria established in Section 2.4. The evaluation is based on the published research work, together with demonstration videos when available, and the associated tools publicly available.

2.7.1 Review

A survey on model-driven engineering tools for developing UIs (Pérez-Medina et al. 2007) covered many existing tools including: ACCELEO, AndroMDA, ADT, AToM3, DSL Tools, Kermet, ModFact, Merlin, MDA Workbench, MOFLON, OptimalJ, QVT Partners, SmartQVT, and UMLX. One of the conclusions made was that these tools are centered on MOF hence support the creation of domain-specific models. However, since these tools are not directly meant for supporting (adaptive) model-driven UIs we shall not consider them as part of our review.

There are some commercial tools that partially support MDE in UI development. However, these tools were not intended for developing adaptive UIs. **Leonardi** is one example; it provides free⁴ and commercial⁵ versions of its application composer. This composer allows designers to visually define the UIs that could be interpreted at runtime. The creators of Leonardi (W4) specify three challenges for business applications: offering high quality user experience, developing software at low cost with minimum technical experience, and providing scalable applications that can accommodate constant business and technological changes.

They claim to face these challenges by supporting MDE agile in Leonardi. This is practically achieved by not generating code from the UI design. Instead, the UI is *interpreted at runtime* through an application engine. We should note that MDE agile is plausible but we noticed some limitations in the way it is applied in Leonardi. Since it is a rapid application development (RAD) tool, Leonardi only supports the concrete UI model and ignores the other *levels of abstraction*. Also, this tool is coupled to a certain extent with Java and does not provide specifications for developing application engines for other technologies. Leonardi is *not intended for developing adaptive UIs* hence it does not offer any tool support for adaptive UI behavior. Other frameworks and tools with fewer

⁴ Leonardi Free: www.leonardi-free.org

⁵ Leonardi Commercial: www.w4.eu

features such as: **OpenXava**⁶ and **Himalia**⁷, provide different model-driven approaches for developing UIs. Yet, the tight coupling of these tools with programming languages (e.g., Java, .NET, etc.) discourages their adoption as a general purpose solution.

Supple (Gajos et al. 2010) partially adopts model-driven UI development hence its tools do not support all the *levels of abstraction*. Basic information on the supporting tools is available in the published work but the tools are not available for the public. According to Gajos⁸, Supple is still a research prototype and he hopes it could be made available for the public in the future.

The **ConcurTaskTrees Environment (CTTE)** (Mori et al. 2002) (version 2.6.3) is a tool for developing and analyzing task models using the CTT notation. CTTE provides a mature UI for designers to devise task models. Yet, it does not provide visual-design tools for all the *levels of abstraction* but it supports the *generation* of the AUI and CUI models in the **MARIA** (Paterno' et al. 2009) language from the CTT task model. The CUI model can be generated as a desktop, mobile, or vocal UI and the final UI can be generated to HTML or Voice XML. However, *synchronization* is not supported in case the CUI changes; also, the generation rules cannot be modified from the tool hence providing a low *predictability* of the generated CUI. MARIA also has a separate authoring environment to separate several *levels of abstraction*. Users can define transformation rules to map the AUI models to CUI models. These rules can be defined through a *visual mapping* between the AUI and CUI model elements. The ability to do so provides a better *predictability* of the generated outcome.

Several tools were presented for supporting the UIDL UsiXML (Limbourg et al. 2004). Some of these tools such as: **UsiComp** (García Frey et al. 2012) and **Xplain** (García Frey et al. 2010), are early-stage research prototypes that provide a limited number of features. The UI models representing the different levels of abstraction are designed in UsiComp inside a single document-style panel. This approach negatively affects the tool's *flexibility* and could prove to

⁶ OpenXava: www.openxava.org

⁷ Himalia: www.bit.ly/HimaliaDotNet

⁸ Krzysztof Gajos on Supple as a Research Prototype: www.bit.ly/SuppleSystem

be tedious when designing UIs for large-scale complex systems. A multi-document *IDE style UI* could be more helpful for developers and IT personnel in managing a large number of artifacts (e.g., UI models, code files, etc.) in real-life software projects.

Similar tools such as: **SketchiXML** (Coyette & Vanderdonckt 2005), **IdealXML** (Montero & López-Jaquero 2007) and **GraphiXML** (Michotte & Vanderdonckt 2008), only target specific phases of the UI construction process hence do not support all the *levels of abstraction*. SketchiXML focuses on transforming manually drawn sketches to concrete UI models. This tool can generate a *predictable* CUI model from the drawn sketches especially if predefined widget sketches were loaded into the system. IdealXML is concerned with modeling task models and generating the abstract UI from them. GraphiXML provides a graphical-design tool for concrete UIs. Even though these tools do not support all the *levels of abstraction*, we consider that they provide a good *control over the UI* since the designer can control the supported models.

Although it is still a limited prototype, UsiComp is the only one of these tools that supports all the *levels of abstraction* and directly targets UI adaptation by applying rules written in the Atlas Transformation Language (ATL) to the UI models. A demonstration showed how these rules could adapt the same UI models to different platforms (web and mobile). The *extensibility of the adaptive behavior* is limited since no clear demonstration is given on how these rules can be extended using the tool. A *visual representation* of these rules is not supported, and the automated *generation and synchronization* between the different levels of abstraction was not demonstrated.

A few supporting tools were presented as part of **MASP** (Feuerstack et al. 2008) including: a *task tree editor* as an Eclipse plugin, in addition to a *layouting tool* and a *task tree simulator* that were offered as standalone tools. The task tree editor can be used in order to model the various tasks, which are required to be supported by a segment of the application being developed. The layouting tool is used for generating layout models. This tool is provided with design models and context-of-use scenarios (device properties and user preferences) as input. The tool provides a box-based layout allowing the designer to specify properties related to containment, order, orientation, and size. However, MASP

lacks a canvas-style visual-design tool for concrete UIs; this could have a negative impact on its *flexibility*. Feuerstack et al. (2006) suggested an HTML WYSIWYG editor to alter the UI; this could prove less useful than a technology independent concrete UI editing tool when targeting multiple presentation technologies. Also, MASP's tools only support adding basic adaptations that are applied to a layout with predefined box-based groupings.

Gummy supports multi-platform graphical UI development (Meskens et al. 2008). It can generate an initial design for each new platform using a combination of features from existing user interfaces. A key objective of Gummy is providing an environment that resembles traditional GUI development tools in order to allow designers to target new platforms without giving up their current work practices. Additionally, Gummy hides the high *levels of abstraction* from the designers, thereby allowing them to operate on the level of abstraction that they are the most familiar with, namely the CUI. Having such designer input on the CUI provides more *control over the UIs*. *Predictable generation and synchronization* is less required because Gummy hides the upper levels of abstraction from the designers. However, some characteristics of the high-level models, such as the temporal operations on CTT tasks models, are not easy to deduce from the CUI. Therefore, it is our belief that it would be better to expose these upper-level models for advanced designers who choose to modify them.

Damask (Lin & Landay 2008) is a tool for prototyping cross-device UIs. It is not intended for developing adaptive UIs but for supporting design-time automatic UI generation. Designers can define a UI layout for one device from which Damask can create an abstract model that it uses to generate the UI layouts for other devices. The employment of patterns in designing the initial UI helps Damask in *generating more predictable* UIs for other devices. The designers can refine the generated UI layouts if necessary, hence providing good *control over the UI*. Both web-style and voice UIs are supported for PCs and mobile phones.

Several criteria were not addressed by the existing tools that we surveyed. Some of these criteria were also not addressed by the techniques we evaluated

in Section 2.6 and include: *extensibility of adaptation types, aspect, and factors*, *empowering new design participants*, and *preserving designer input on the UI*.

We should note that most of the surveyed tools are intended for developing model-driven UIs but do not support adaptation capabilities. Therefore, the *extensibility of the adaptive behavior* and the definition of *visual and code-based adaptive behavior* are only partially supported by MASP, UsiComp, and MARIA. Also, apart from Leonardi, the tools do not provide a mature *IDE style UI* that could ease the development process.

In spite of the heterogeneity in the types of platforms (e.g., desktop, web, mobile) for which UIs can be generated by some tools such as: Damask and MARIA, the generated UIs only follow the WIMP style. Therefore, the tools we surveyed do not demonstrate a high level of *completeness* since their ability to support other UI types such as multi-touch tabletop UIs was not demonstrated.

Most of the tools we surveyed provide a visual-design canvas for the models they support. Therefore, we considered that they fulfill the *expressive match* criterion. We considered tools such as: Damask, Gummy, and Leonardi, which support a form of integrated testing to fulfill the *flexibility* criterion. Besides Leonardi, the tools we surveyed do not support reusability of UI model parts (e.g., in the same way visual components are reused in traditional IDEs). Also, reusability of adaptive behavior is not demonstrated although it could be possible in principle in tools such as: MARIA, MASP, and UsiComp, which support transformation rules. Hence, we only considered these few tools to partially fulfill the *expressive leverage* criterion.

Achieving a low *threshold* and a high *ceiling* is noted to be one of the major criticisms regarding tools supporting model-driven UI development. Therefore, building models graphically was suggested to achieve a lower threshold (Vanderdonckt 2008). We can say that Damask, Gummy, and Leonardi potentially have a lower threshold than other tools since they promote a development technique that starts with the CUI similar to the techniques adopted by classic IDEs, which are more familiar to designers. In terms of achieving a high ceiling, since most of the tools are research prototypes, it is hard to consider them as alternatives for commercial IDEs (e.g., Visual Studio, Eclipse, etc.) that can be used to develop real-life working commercial software

applications. One exception is Leonardi, which is already a commercial IDE that is used for developing working software applications. Yet, Leonardi does not support adaptive behavior. Hence, we considered the surveyed tools to partially fulfill the *threshold and ceiling* criterion.

2.7.2 Summary of the Review

We present a comparison of the tools we reviewed in Table 2.5. The criteria that were not addressed by the existing techniques are: *preserving designer input on the UI*, *empowering new design participants*, and *extensibility of adaptation types, aspects, and factors*. Other criteria that have major gaps include providing the ability to *define and extend adaptive behavior both visually and through code*, better *expressive leverage*, an *IDE style UI*, more *completeness*, and improving the *threshold and ceiling*.

Table 2.5: Visual Evaluation and Comparison of Adaptive Model-Driven User Interface Development Tools

Legend														
	Completeness	Control over the UI	Extensibility of adaptive behavior	Ext. of adaptation types, aspect, factors	Empowering new design participants	IDE style UI	Levels of abstraction	Predictable modeling, generation, and sync.	Preserving designer input	Flexibility	Expressive leverage	Expressive match	Threshold and ceiling	Visual and code-based adaptive behavior
CTTE (HIIS Lab)	◐	●	○	○	○	◐	◐	◐	○	◐	○	●	◐	○
Damask	◐	●	○	○	○	◐	◐	●	○	●	○	●	◐	○
GraphiXML	◐	●	○	○	○	◐	◐	○	○	◐	○	●	◐	○
Gummy	◐	●	○	○	○	◐	●	●	○	●	○	●	◐	○
IdealXML	◐	●	○	○	○	○	◐	◐	○	◐	○	●	◐	○
Leonardi	◐	●	○	○	○	●	◐	○	○	●	◐	●	◐	○
MARIA (Tools)	◐	●	◐	○	○	◐	●	●	○	◐	◐	●	◐	◐
MASP (Tools)	◐	●	◐	○	○	◐	◐	◐	○	◐	◐	●	◐	◐
SketchiXML	◐	●	○	○	○	○	◐	●	○	◐	○	●	◐	○
UsiComp	◐	●	◐	○	○	○	●	◐	○	◐	◐	●	◐	◐

2.8 Chapter Summary

The literature includes many systems that offer solutions for developing adaptive UIs in an attempt to address software usability problems. The model-driven approach formed the basis of most of these systems. This review presented an overview and evaluation of existing adaptive model-driven UI development systems. We established a set of criteria by either directly drawing on recommendations from the literature or indirectly from features dispersed in multiple existing systems. We classified the related systems under reference architectures, practical techniques, and supporting tools, and evaluated them according to the criteria we established. We tabulated the result to illustrate the level to which each criterion is fulfilled by each of the surveyed systems.

After reviewing the reference **architectures** for developing adaptive model-driven UIs, we found that there are still gaps and room for improvement in the following criteria: *user feedback on the adapted UI*, *trade-off analysis*, *integrating in existing systems*, *empowering new design participants*, and adopting a *modeling approach* that uses interpreted runtime models. We offer improvements in our Cedar Architecture (**Chapter 4**), which promotes the use of interpreted runtime models for handling more advanced adaptations. It also provides a high level description for supporting user feedback and trade-off management. The Cedar Architecture also supports the integration in existing enterprise applications and empowers non-technical stakeholders to participate in the adaptation process.

We found a major gap in **feature-set adaptation techniques** that suffer from one or more of the following problems: lack of a practical implementation mechanism, lack of generality, or restriction to design-time adaptation without offering a runtime adaptive solution. Our attempt to fill this gap is presented by the Role-Based User Interface Simplification (RBUIS) mechanism (**Chapter 5**). RBUIS is a general-purpose, model-driven, and tool supported adaptive UI mechanism capable of reducing UI bloat at runtime by adapting the UI's feature-set based on the context-of-use.

We found several gaps in **layout optimization techniques** including: *preserving designer input on the UI*, *empowering new design participants*,

integrating in existing systems, and extensibility of adaptation types, aspects, and factors.

Some approaches attempted to address the need of *preserving designer input on the UI*. Smart templates were proposed for improving automatic generation of ubiquitous remote control UIs on mobile devices (Nichols et al. 2004). Although these templates improve the ability to preserve designer input, specifying the various template variations could be time consuming and would be classified under adaptable rather than adaptive behavior. Raneburger et al. (2012), attempt to enhance the quality of generated UIs by using a graphical tree editor to add hints to model transformations (e.g., the alignment of a widget). However, UI designers might only work on the CUI level and leave the model transformations to the developers. Also, the authors state that a graphical WYSIWYG editor would improve on their approach. Hence, we present a technique (Akiki et al. 2013c) for preserving designer input by allowing UI designers to add constraints on the CUI model. However, this point is only partially addressed by the research that we have done so far and its completion is left for future work.

A few approaches have tried to *empower new design participants* in the UI adaptation process through the means of crowdsourcing. Adaptable Gimp (Lafreniere et al. 2011) was presented as a socially adaptable alternative of the GNU image manipulation tool, Gimp. It allows the community to customize its UI by creating task-sets in a wiki. Another approach (Nebeling et al. 2012) allows HTML-based UIs to be adapted by end-users through a toolkit with a predefined set of adaptations. The changes are stored in a central repository as Cascading Style Sheets (CSS), which could be applied for other end-users with similar needs. However, these approaches are not model-driven hence making them technology-dependent and only focus on end-user manual adaptation. Therefore, we presented an approach that complements our RBUIS mechanism by engaging end-users to help technical stakeholders in defining the adaptive UI behavior using a simple online tool (Akiki et al. 2013b). In order to set a boundary for this thesis, we only partially addressed this point in our research and left its completion for future work.

As far as integrating in existing systems and extensibility of adaptation types, aspects, and factors, no works were presented to thoroughly target these points. We demonstrate the ability of our approach to address these points in **Chapter 5** and **Chapter 7**. Other criteria such as: *trade-off analysis*, *user feedback on the adapted UI*, and *visual and code-based adaptive behavior representation* were partially addressed by the existing art and could be improved upon. Some techniques exist for improving the visual representation of adaptive UI behavior (adaptation rules). A graphical tool was suggested for hiding the complexity of defining UI adaptation rules (López-Jaquero et al. 2009). However, this tool might not be able to handle all possible scenarios due to the limited use of a high level visual representation. As part of our RBUIS mechanism (**Chapter 5**), we presented a technique that supports the definition of visual and code-based adaptive behavior by employing workflows. These workflows support visual programming constructs that can be extended as needed such as: control structures, error handling, etc. Also, it is possible to define code-based adaptation operations in scripts that integrate in the workflows.

We consider **tool support** to be crucial for the adoption of adaptive model-driven UI development by the software industry. However, the present tools still require a lot of work before they can become comparable to existing industrial quality integrated development environments such as: Visual Studio.NET and Eclipse. In addition to the gaps they share with adaptive UI techniques, there are several points in the existing tools that need to be improved such as: the *extensibility of the adaptive behavior*, *expressive leverage*, and *threshold and ceiling*. Several of the existing tools such as: Damask and Gummy, have a good starting point. Separate tools coming from the same research groups could be merged together such as: GraphiXML, IdealXML, and SketchiXML on one hand, and MARIA and CTTE on the other hand. This merger will make these tools more comprehensive before new enhancements can be added. We presented our own tool called Cedar Studio (**Chapter 6**) to provide a unified IDE, which supports the development of adaptive model-driven UIs. It supports visual-design tools for: task models, domain models, AUI models, CUI models, and workflows that represent the adaptive behavior. It also supports code-editing tools for adaptive behavior scripts and model checking constraints. Furthermore, Cedar Studio supports automatic generation

and synchronization between the various levels of abstraction (task, AUI, and CUI models) with the possibility to make manual changes at any of these levels.

The next chapter establishes the research questions and hypotheses based on the gaps determined by the literature review presented here. Furthermore, the research methods for answering the questions are discussed and justified.

3

Research Design: Research Questions, Hypotheses, and Methods

“If we knew what it was we were doing, it would not be called research.”

— Albert Einstein

This chapter starts by dividing the overarching research question it into sub-questions and explains them. Afterwards, the hypotheses statements are established based on the research questions. Then, the selected research methods are discussed and justified. An overview is presented of the existing research and technologies that were used as a basis for parts of our work. Finally, the chapter is summarized and an overview of the research steps is given.

3.1 Research Questions

Easterbrook et al. (2008) differentiate between “*knowledge*” and “*design*” questions in software engineering research. They note that knowledge questions focus on “*the way the world is*”, whereas design questions tend to focus on establishing “*better ways to do software engineering*”. Empirical research is usually the path chosen by researchers posing knowledge questions as opposed to an engineering approach taken by researchers with design questions.

Our research follows an engineering approach with a mixture of both design and knowledge questions. The design questions are answered by devising an approach that supports the development of adaptive enterprise application UIs. On the other hand, the knowledge questions are answered by evaluating this approach empirically from the technical and human perspectives.

In Chapter 1, we identified the following overarching research question:

How can adaptive user interfaces be leveraged for improving the usability of enterprise applications?

In this chapter, we shall divide the overarching question into sub-questions related to the novel technical contributions and their evaluation. These sub-questions are based on the results of the gap analysis, which we conducted as part of the literature review in Chapter 2.

3.1.1 Technical Contribution Research Questions

We define the following sub-questions that are related to the novel technical contributions of this thesis:

Q1: What reference architecture can guide the development of adaptive enterprise application UIs based on a runtime model-driven approach, while supporting: user feedback, trade-off analysis, integration in existing systems, and the empowerment of non-technical design participants?

The proposed reference architecture includes a high level description of the architectural components, their distribution among the architecture's layers, and the way they communicate with each other.

Q2: What UI adaptation technique, based on the proposed reference architecture (Q1), can minimize the feature-set and optimize the layout of enterprise application UIs according to the context-of-use for an extensible number of adaptation aspects and factors, and support the addition of both visual and code-based adaptive behavior as needed?

To answer this question, we provide design specifications for the adaptation technique. These specifications include meta-models, algorithms, and a means for representing the adaptive behavior visually and through code. To make the offered solution general purpose, an application programming interface (API) is devised to support building enterprise application UIs, which can use the adaptation technique.

Q3: What trade-off management mechanism can support multi-aspect trade-off analysis that works with the devised UI adaptation technique (Q2)?

The proposed trade-off management mechanism is able to analyze the trade-offs among multiple conflicting adaptation aspects (e.g., screen-size, physical impairments, etc.) by using a quantitative measure that determines the quality of each UI design. Furthermore, this mechanism is able to accommodate multiple adaptation aspects that can be extended as needed.

Q4: What integrated development environment can help software developers and IT personnel in developing and maintaining model-driven enterprise application UIs and adapting them with the devised adaptation technique (Q2)?

The IDE in question provides the necessary visual-design and code-editing tools for supporting the creation and maintenance of the artifacts required for developing adaptive model-driven enterprise application UIs. These artifacts primarily include the UI models and adaptive behavior (adaptation rules). A basic early prototype of this IDE supporting a visual-design tool for a single type of UI model (concrete UI) is demonstrated in Figure 3.1. We built this prototype as part of an early exploration of this research work (Akiki 2010).

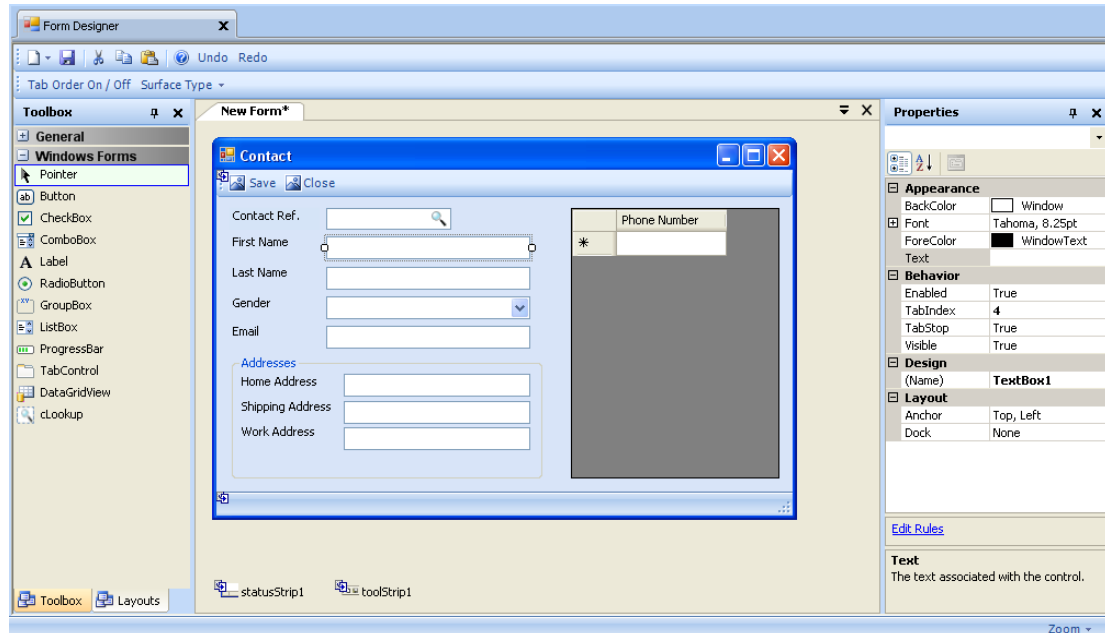


Figure 3.1: An Early Prototype of Our IDE (Akiki 2010)

3.1.2 Evaluation Research Questions

We define the sub-questions related to the evaluation of the novel technical contributions of this thesis as follows:

3.1.2.1 Software Engineering Perspective

Q5: Does the devised UI adaptation approach (Q1 and Q2) integrate in existing enterprise applications without causing major changes to the way they function or incurring a high integration cost?

To answer this question, we measure the cost of reverse-engineering existing non-model driven enterprise application UIs into a model-driven representation. We also measure the integration change impact in terms of the lines-of-code, which have to be added or changed.

Q6: Does the proposed UI adaptation technique (Q2) provide a real-time runtime performance and is it scalable?

Runtime performance is measured using a metric that is composed out of multiple time components, which represent the efficiency of different parts of the UI adaptation technique. The scalability of this technique is determined through a complexity analysis of its algorithms, and by load-testing its implementation to simulate a real-life enterprise application workload.

Q7: What is the perspective of industry experts on the generality and flexibility of the devised UI adaptation technique (Q2)?

This question can be answered by obtaining the perspectives of industry experts after showing them videos of our adaptation technique in operation, the way it integrates into existing systems, and its supporting tool.

3.1.2.2 Human-Computer Interaction Perspective

Q8: Does feature-set minimization and layout optimization significantly improve the usability (efficiency, effectiveness, and satisfaction) of enterprise application user interfaces?

Usability studies are conducted to answer this question. The participants are asked to perform tasks using two versions of the same UI, an initial version and an adapted one. Several measurements are used to evaluate whether the adapted UIs significantly improves usability. The time taken by each participant to complete the tasks is recorded to measure the efficiency. Eye-tracking is used to measure how lost the participants are when using complex UIs versus the simplified ones. The effectiveness is evaluated by measuring the number of mistakes each participant made when completing the task in the different UI versions. Participants are asked to provide feedback on their satisfaction (perceived usability) by answering a set of questions.

3.2 Hypotheses

We do not define any hypotheses for the technical contribution research questions from Section 3.1.1 since these questions will be answered by devising novel solutions (architecture, technique, and tool). However, we define the following four null hypotheses for the evaluation research questions from Section 3.1.2, since these questions are answered through empirical studies:

H₀₋₁: The devised UI adaptation approach cannot integrate in existing enterprise applications without causing major changes to the way they function or incurring a high integration cost.

H₀₋₂: The proposed UI adaptation technique does not provide real-time performance (milliseconds) and is not scalable for complex problems.

H₀₋₃: Industry experts will not find our approach general and flexible enough to be used in real-life projects.

H₀₋₄: Minimizing the feature-set and optimizing the layout of enterprise application UIs based on the context-of-use, does not significantly improve their usability (efficiency, effectiveness, and satisfaction).

3.3 Research Questions for Future Work (Partially Addressed)

The literature review we conducted in Chapter 2 identified several gaps in the state-of-the-art. However, completely addressing all these gaps is out of the scope of one thesis. Therefore, we only address the following two sub-questions partially and leave the remaining parts for future work.

Q9: What technique can empower non-technical stakeholders such as end-users to contribute to the UI adaptation process?

The proposed technique should allow non-technical stakeholders to contribute to the UI adaptation process by defining adaptive behavior using tools that were particularly developed for people with basic or no technical skills.

Q10: What can be added to the UI adaptation technique (Q2), for allowing UI designers to preserve some of their input on the UI after it is adapted?

To answer this question, a technique must be devised to allow UI designers to add design-time input, on the UI models, which gets preserved even after the UI is adapted. The added input would embody the characteristics of the UI that require human ingenuity and are not met by automated adaptation techniques.

3.4 Research Methods

Several methods can be applied for conducting and evaluating software engineering and human computer interaction research. For example, the most common kinds of validation in software engineering research (based on submissions to the international conference on software engineering) are: systematic analysis and experience in actual use (Shaw 2002). This section discusses both the engineering techniques and the empirical research methods, which we applied to answer the research questions.

3.4.1 Engineering Techniques

Several engineering techniques are employed in this thesis to answer the technical contribution (design) research questions listed in Section 3.1.1. The outcome is evaluated empirically by answering the evaluation questions listed

in Section 3.1.2. The engineering techniques include meta-modeling, devising algorithms, and implementing support tools and prototypes. Such techniques are not uncommon when researching a topic that offers a software engineering solution for a human computer interaction problem. Other researchers working in a similar area, for example Blumendorf (2009), Feuerstack (2008), and Florins (2006), contributed meta-models, support tools, and prototypes that were evaluated using worked examples.

3.4.2 Empirical Methods

The empirical methods are employed for both exploratory and confirmatory purposes. A survey is used for an initial exploration, whereas worked examples and software metrics, interviews, and controlled experiments were used for evaluating the outcome.

3.4.2.1 Surveys

We used a survey in the form of an online interactive questionnaire for the initial investigation study, which we presented in Section 1.3.4. A survey could identify the characteristics of a broad population of individuals if a clear research question inquiring about the nature of the target population is present (Easterbrook et al. 2008). This precondition is available in the study that we conducted. By depending on a representative sample we were able to test for statistical significance. Furthermore, when compared to other data collection techniques, questionnaires can reach many people by employing a low amount of resources (Sharp et al. 2007).

3.4.2.2 Worked Examples and Software Metrics

Easterbrook et al. (2008) identify several research methods, which include case studies. The authors mention that there is confusion about this term by using it in the sense of a worked example, whereas as an empirical method it should be used as an inquiry. We use worked examples in the form of enterprise application prototype UIs, which adapt to the context-of-use.

By defining and applying software engineering metrics to the worked example prototypes, we can evaluate how well our approach integrates in existing systems (Q5) and whether it is scalable and provides real-time runtime performance (Q6).

3.4.2.3 Controlled Experiments

A controlled experiment can be used to test a hypothesis by manipulating one or more dependent variables and measuring their effects on one or more independent variables (Easterbrook et al. 2008).

Usability studies are common in human computer interaction research. They can be used to measure the difference in usability between one UI design and another. Participants can be asked to perform the same task using two different adaptations of the same interface. The participants' efficiency, effectiveness, and satisfaction can be measured for each UI version. A comparison between the measurements validates if feature-set minimization and layout optimization can significantly improve the usability of enterprise applications (Q8).

3.4.2.4 Interviews

Interviews are a form of survey research. However, we do not intend to use this method with a broad population but with a limited number of industry experts in order to answer research question Q7. We rely on a semi-structured interview to ask about the generality and flexibility of our adaptation technique, while allowing the interviewees to maintain a broader scope that allows us to explore additional insights.

3.5 Building on Existing Research and Technologies

Our research provides novel contributions. However, as a good practice it makes use of the existing literature and technologies, which saves the time of having to devise everything from scratch.

We build upon work presented in existing reference architectures and frameworks such as: the CAMELEON reference framework and the Three

Layer Architecture (Section 2.3.2). These research works provide a theoretical basis for our proposed reference architecture.

We also make use of existing technologies. For example, the concrete UI model visual-design tool offered by our IDE is based on the Windows Forms visual-design component, which is part of the .NET framework. This component saved us the effort of having to create the tool completely from scratch. In order to represent the adaptive behavior for layout optimization, we rely on the Windows Workflow Foundation (WF), which is provided as part of the .NET framework. The visual-design tool for these workflows is incorporated in our IDE and provides the means for defining the adaptive behavior.

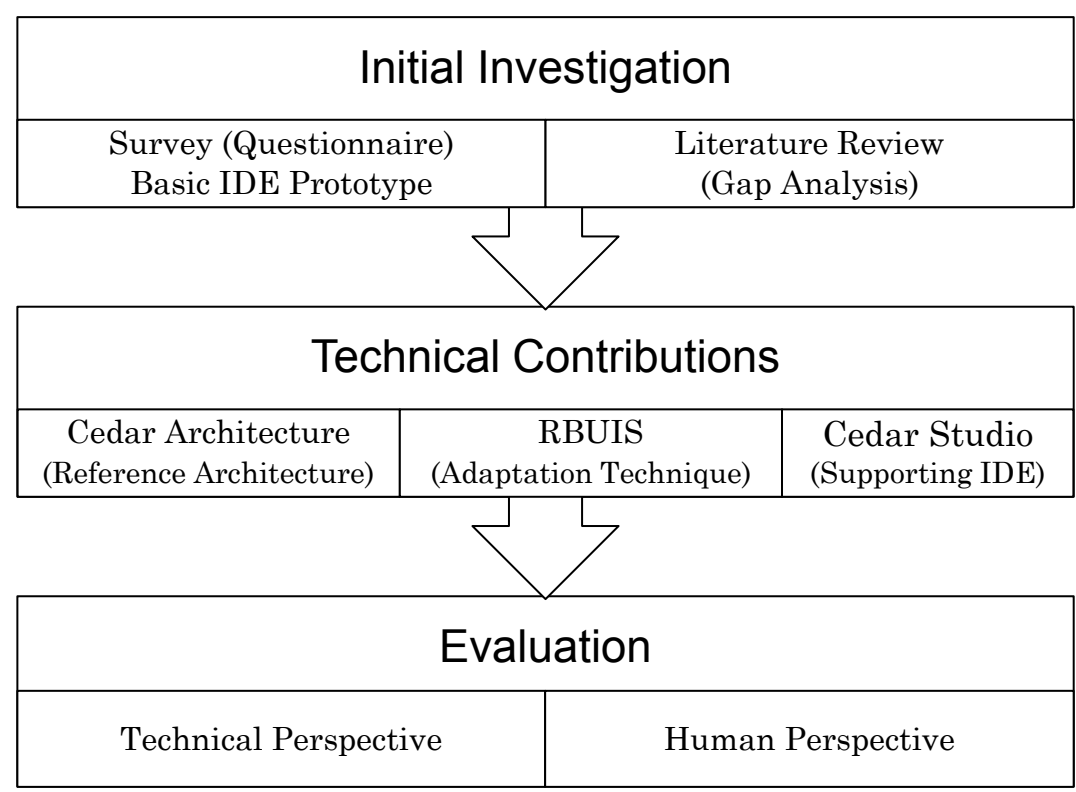


Figure 3.2: Overview of the Research Steps

3.6 Chapter Summary

In this chapter, we divided the overarching research question into sub-questions and provided an explanation for each one. These sub-questions inquire about the thesis’ technical contributions and their evaluation. We established our hypotheses and discussed the research methods, which we used

to answer the research questions. We also presented an overview of the existing research and technologies that are used as a basis for parts of our work.

An overview of the steps undertaken in this thesis, including the initial investigation, technical contribution, and evaluation, are illustrated in Figure 3.2. As we mentioned in Section 1.4 of Chapter 1, this thesis makes the following three novel contributions:

- **Cedar Architecture:** A reference-architecture for supporting the development of adaptive model-driven enterprise application UIs.
- **RBUIS:** A UI adaptation mechanism based on the Cedar Architecture for simplifying enterprise application UIs by minimizing their feature-set optimizing their layout according to the context-of-use.
- **Cedar Studio:** An IDE for supporting different stakeholders such as: software developers and IT personnel, wishing to use RBUIS.

These contributions are evaluated empirically from the technical and the human perspectives. The evaluation covers both the software engineering and human computer interaction areas.

4

The Cedar Architecture: A Reference for Developing Adaptive Model-Driven Enterprise Application User Interfaces

“The greatest products of architecture are less the works of individuals than of society; rather the offspring of a nation's effort, than the inspired flash of a man of genius...”

— Victor Hugo, *The Hunchback of Notre-Dame*

This chapter introduces the layers and components that constitute the Cedar Architecture, which serves as a reference for stakeholders interested in developing adaptive enterprise application UIs based on an interpreted runtime model-driven approach. A general-purpose meta-model is also introduced as a high-level design for supporting the development of adaptive UIs based on the Cedar Architecture. Finally, we report the results of an experiment, which demonstrated that user interfaces developed using interpreted runtime models, can be loaded and rendered as efficiently as those based on compiled code.

4.1 Introduction

Kramer & Magee (2007) promote the use of an architecture-based approach for devising adaptive software systems since it could build on existing work and provides: generality, a level of abstraction, potential for scalability, and potential for integrating work from multiple areas (e.g., modeling and representation, analysis, etc.). The existence of reference architectures for adaptive user interfaces could help in realizing these UIs in complex software systems such as enterprise applications.

In Chapter 2, we identified several gaps in the existing state-of-the-art reference architectures. These architectures did not adopt a *modeling approach* that uses interpreted runtime models for offering more flexibility in terms of realizing advanced UI adaptations. Additionally, existing architectures do not support *end-user feedback* on the adapted UI and *trade-off analysis* among a varying number of conflicting adaptation aspects. Furthermore, the existing works did not demonstrate how adaptive UI techniques, which are based on their proposed architectures, can *integrate in existing software systems*. Finally, the existing architectures do not offer a solution for *empowering new design participants* such as end-users, by offering them the ability to take part in defining the adaptive user interface behavior.

In this chapter, we introduce the Cedar Architecture, which serves as a reference for stakeholders interested in developing adaptive enterprise application UIs based on a model-driven approach. Our architecture was designed to fill the gaps that were identified in Chapter 2. It promotes the use of interpreted runtime models, which allow UIs to be loaded, adapted, and rendered dynamically without resorting to code generation. Additionally, it offers high-level components for supporting functionality such as: end-user feedback and trade-off analysis. The Cedar Architecture is service-oriented hence it promotes a separation of concerns between the enterprise applications that require UI adaptation and the adaptive UI technique that provides this capability. Our architecture's client-side layer has an application programming interface (API), which allows UI adaptation techniques to integrate in existing enterprise applications. These applications would gain adaptive UI capabilities by calling web-services through the API to access the UI adaptation technique components on the server-side layers.

The Cedar Architecture is meant to answer research question Q1, which we established in Chapter 3 as follows:

Q1: What reference architecture can guide the development of adaptive enterprise application UIs based on a runtime model-driven approach, while supporting: user feedback, trade-off analysis, integration in existing systems, and the empowerment of non-technical design participants?

4.2 The Cedar Architecture

This section provides a high-level overview of the components comprising the Cedar Architecture, which is based on the: CAMELEON reference framework (UI abstraction), Three Layer Architecture (adaptive system layering), and Model-View-Controller (MVC) paradigm (implementation). As we discussed in Chapter 2, CAMELEON provides a reference for representing UIs on multiple levels of abstraction and the Three Layer Architecture provides guidance for layering adaptive software systems. Basing the Cedar Architecture on these works, allows it to cover both model-driven UI representation and adaptive behavior. Furthermore, MVC offers implementation guidelines that promote a separation of concerns between the UI, the domain model, and the implementation code. The layers and components comprising the Cedar Architecture are illustrated in Figure 4.1 and explained in Sections 4.2.2 and 4.2.3. The steps of the adaptation procedure are shown sequentially (S1 to S5) in Figure 4.1 and are explained in Section 4.2.4.

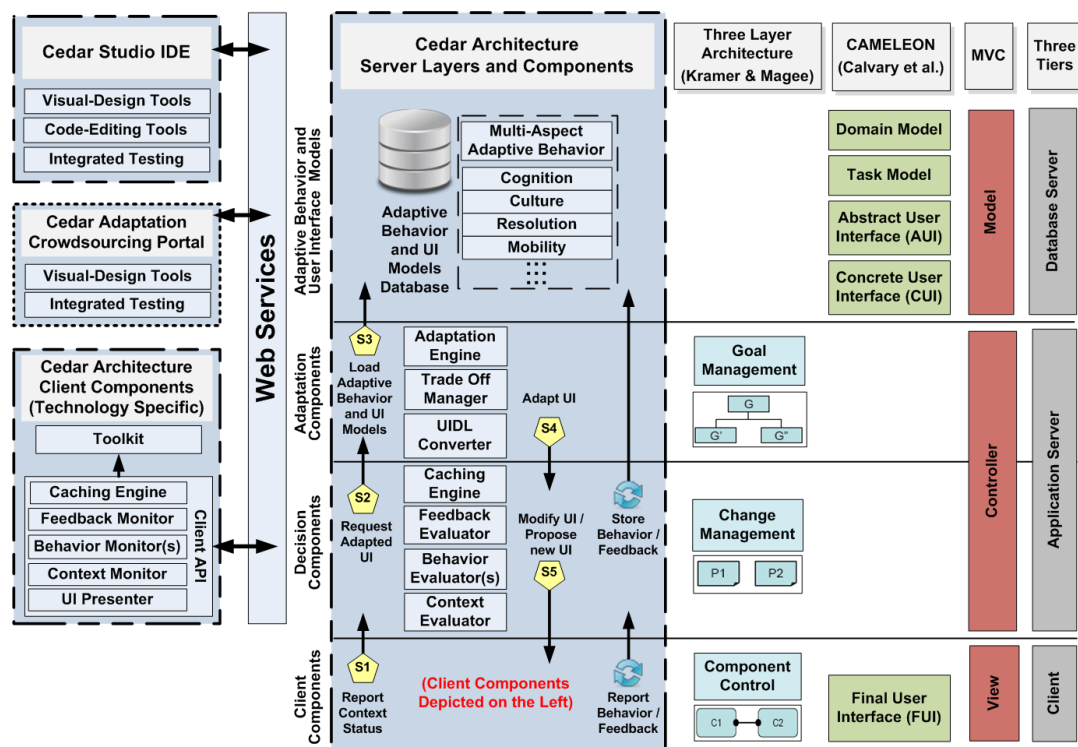


Figure 4.1: The Cedar Architecture

4.2.1 Using Interpreted Runtime Models

The Cedar Architecture adopts the model-driven approach for developing adaptive user interfaces. Runtime models are usually more desirable for developing adaptive UIs since they are dynamic in nature and hence can be used to adapt the UI while the software application is running. However, in certain scenarios, using runtime models while maintaining the generated code-based artefacts is insufficient for achieving the required adaptive behavior.

One example scenario is an adaptation operation, which performs elimination, substitution, and realignment of user interface widgets. The adaptation could, for example, eliminate a subset of the functionality that is unnecessary for a certain end-user. Then, the UI could be reshaped according to this end-user's computer literacy level as describe in the following hypothetical example:

- Beginner: The user interface is presented in wizard form, since a series of small steps are easier to interpret.
- Intermediate: The user interface is divided among several tabs.
- Expert: The user interface widgets are displayed on one page.

An adaptive UI approach that aims to fulfill the previously mentioned example should provide runtime support for actions such as: eliminating widgets, replacing a widget with another, or adding new widgets that did not previously exist during the development phase. Performing such actions at runtime would be difficult when the UI is based on generated artefacts. For example, substituting a widget with another is difficult since the widget types are hard-coded, whereas with runtime interpretation, the types could be switched in the model and the widget would be rendered accordingly.

Our approach uses interpreted runtime models hence there is no need to generate code for creating the UI. Instead, the models are interpreted at runtime, adapted according to the context-of-use, and presented to the end-user as a running user interface.

4.2.2 Layers Comprising the Cedar Architecture

The Cedar Architecture comprises a client-side layer called client components, and three server-side (application / database servers) layers: decision components, adaptation components, and adaptive behavior and UI models.

Following the Three Layer Architecture (Kramer & Magee 2007), the adaptation components layer of the Cedar Architecture performs the Goal Management, its decision components layer handles Change Management, and its client components layer performs Component Control. Following the MVC paradigm, models are represented on the adaptive behavior and UI models layer, the Controller spans the adaptation components and decision components layers, and the View is managed by the client components layer.

The server-side layers are accessed through web-services from the client components layer, our IDE Cedar Studio (Chapter 6) and our crowdsourcing portal (Akiki et al. 2013b). Cedar Studio supports stakeholders such as: software developers and IT personnel, in defining and maintaining UI models and adaptive behavior. The crowdsourcing portal on the other hand, empowers non-technical stakeholders like end-users, by allowing them to participate in defining adaptive UI behavior.

This section provides a general explanation of the role of each of the components comprising the client-side and server-side layers of the Cedar Architecture. The coming chapters provide more implementation information. A UI adaptation technique, based on the Cedar Architecture, is presented in Chapter 5. It provides more details on the way we implemented the architectural components. In this chapter, our IDE Cedar Studio and crowdsourcing portal are merely illustrated as part of Figure 4.1 to show how they fit within the overall architecture. Nonetheless, Cedar Studio is presented in Chapter 6 while details on the crowdsourcing portal can be found in a separate paper (Akiki et al. 2013b).

4.2.2.1 Client Components Layer

The components in this layer are deployed to the client machine and are the only technology-specific components in the Cedar Architecture. Since these

components are part of the application programming interface (API) and have to be integrated in the enterprise application's code, a different version is required for each programming language and presentation technology.

The *Context Monitor* is responsible for monitoring any changes to the triplet forming the context-of-use: user, platform, and environment. For example, it can monitor if there is a change in the: end-user's role(s), device's screen-resolution, distance between the end-user and the display presenting the UI, etc.

Adaptive mechanisms could affect an end-user's UI control (McGrenere et al. 2002). End-users could feel loss of control if the adaptive mechanism makes decisions they cannot understand or change. Reduction mechanisms could affect feature-awareness (Findlater & McGrenere 2007). If a UI was adapted by reducing functionality without providing a means of exploring the features that were removed and possibly bring them back, the end-users could become unaware of some features that they might want to use in certain contexts. These negative effects could be overcome if the end-users are kept in the adaptation loop by allowing them to provide feedback on the UI adaptation operations performed by the system. The *Feedback Monitor* allows end-users to report their feedback on the UI adaptations performed by the system and offers the ability to reverse adaptations or choose other alternatives when possible.

An important part of any dynamic approach is data caching. The ability to cache data on the client-side, allows UIs to be dynamically rendered more efficiently. The *Caching Engine* is responsible for caching the UI models for allowing interpreted runtime models to have the performance of compiled code. Caching provides the robustness required by enterprise applications without neglecting the ability to customize such applications at runtime.

The *UI Presenter* is responsible for interpreting the adapted UI models and presenting a running UI to the end-users. In the case of graphical UIs, which are the most common in enterprise applications, the UI Presenter renders the UI model using an existing presentation technology (e.g., HTML). Theoretically, this component is responsible for handling data-binding, event management, and validation by linking the dynamically created UI to the application's code-behind and domain-model. However, we did not implement this part of the component since our work is more concerned with UI adaptation and presentation.

4.2.2.2 Decision Components Layer

These components are deployed to the application server to handle decision making in various adaptive UI scenarios.

The *Context Evaluator* handles the information submitted by the *Context Monitor*, in order to evaluate whether the change requires the UI to be adapted. For example, an end-user holding a mobile phone could suddenly start walking at a faster pace. In this case, the Context Monitor can use the phone's accelerometer to detect this new pace and report it to the Context Evaluator, which checks whether the walking speed requires the UI to be adapted.

The *Caching Engine* on the application server assumes a role similar to that of its counterpart on the client machine. However, in this case, the caching is not done on the session level for each individual end-user but on the application level for all the end-users. The UI models cached at this level would have already been adapted. Hence, in case the same adaptation is required by a different end-user, the models would be loaded from the cache rather than re-performing the adaptation, which could be more time consuming.

4.2.2.3 Adaptation Components Layer

The components in this layer are deployed to the application server in order to handle the adaptation of the UI models.

The *Adaptation Engine* is responsible for adapting the UI models by executing the appropriate (based on the context-of-use) adaptive behavior on them.

Enterprise application UIs can be adapted according to multiple adaptation aspects. Hence, trade-off management is necessary in certain situations where conflicting aspects make it impossible to fully meet all the constraints. In such situations, the *Trade-off Manager* assumes the role of managing the trade-offs between conflicting adaptation aspects that affect the same user interface.

The adapted UI is transmitted to the client machine in an XML format to permit adaptation techniques based on the Cedar Architecture to be consumed as a generic service through an API from different enterprise applications. The

format could be one of the known UI description languages such as: UsiXML, UIML, etc. The *UIDL Converter* is responsible for handling the conversion between the UI model (stored as relational data) and the selected UIDL format. Alternatively, the UIDL Converter could simply convert the UI models into an XML document based on our own representation. We chose the latter approach because this work is concerned with UI adaptation more than representation.

4.2.2.4 Adaptive Behavior and UI Models Layer

This layer hosts the adaptive behavior and UI models, which are stored in a relational database on a database server. The relational database serves as a common repository and makes it easier to manage these artifacts at runtime using Structured Query Language (SQL) operations.

The adaptive behavior is a generic set of rules according to which the UI can be adapted when the context-of-use changes. Such rules could be based on various adaptation aspects. The adaptive behavior can be applied on any of the UI models representing the different levels of abstraction: task, domain, abstract UI, and concrete UI. The way we represent the adaptive behavior is explained in Chapter 5 as part of our UI adaptation technique. As for the UI models they conform to the meta-model shown in Figure 4.4 and explained in Section 4.3.2.

4.2.3 Adaptive Behavior Data: How Should the UI be Adapted?

The data that is used for making decisions about the ways of adapting the UI could be obtained from a variety of sources such as: *context-models* that are based on empirical studies (e.g., Section 1.3.4) or expert knowledge, *end-user feedback* that is obtained through the Feedback Monitor, and *monitoring behavior change* that is performed by the Behavior Monitor(s).

Adaptive behavior data is defined in *context-models* that are stored in a relational database on the database server. These models can represent adaptive behavior for different adaptation aspects pertaining to any context-of-use pillar. Empirical studies could be conducted within an enterprise to identify how the UI should be adapted for particular adaptation aspects such as

computer literacy. On the other hand, expert knowledge could be enough to define these models for other adaptation aspects such as different screen-sizes.

Context-models can be refined through *end-user feedback* that is collected on the Client Components Layer through the *Feedback Monitor*. End-user feedback is reported to the Decision Components Layer to be evaluated by the *Feedback Evaluator* before it is stored in the database as a refinement for the context-models. An example of end-user feedback could be: Choosing to show features that were removed from the UI, or making alternative adaptation choices like selecting to group the UI widgets using tab-pages rather than group-boxes. In the example of showing removed features, the Feedback Evaluator could check whether showing a removed feature requires other features to be shown as well due to interdependency.

New situations and behavior changes are detected on the Client Components Layer by the *Behavior Monitor* and are reported to the Decision Components Layer in order to be evaluated by the *Behavior Evaluator*. Some examples of what could be monitored include: the end-users' usage rate of input fields, new updates installed on the platform, information collected from the environment through sensors, etc. For example, an end-user could be initially allocated access to part of the UI's features. Yet, the behavior change monitor could detect that even in this part of the UI there are unused fields hence triggering an update to the adaptive behavior data to indicate that these fields should be removed as well. The Behavior Evaluator could evaluate, for example, whether the usage rate of a certain field is low enough to exclude it from the UI.

4.2.4 Adaptation Procedure

Some systems such as: MASP (Blumendorf et al. 2010) and MyUI (Peissner et al. 2012), directly adapt UIs while the end-user is working (direct adaptation) due to the ubiquitous nature of their target applications. On the other hand, McGrenere et al. (2002) promote offering the adapted UI as an alternative version (indirect adaptation) because direct adaptation could confuse the end-users. We think that both approaches are necessary to cater for a wider variety of adaptations. For example, if an end-user is working on a mobile phone while

sitting down, and then suddenly stands up and begins to walk fast, the UI could be adapted directly to cater for this change in the context-of-use. On the other hand, if the UI requires adaptation to each end-user's computer literacy level, this level could be known and stored in the enterprise application database in advance. Hence, when the end-users log into the application, they will be given access to an adapted version of the UI that meets their particular profile.

Since enterprise applications mostly contain WIMP-style UIs, which are used in an office environment, proposing the adapted UI version as an alternative could be a better adaptation choice. However, our architecture supports both direct and indirect adaptation to also cater for UIs, such as the ones running on mobile phones, which have to directly adapt to an evolving context-of-use.

The adaptation procedure is shown as a part of the architecture in Figure 4.1 by steps S1 to S5. These steps could be mapped to the flow chart in Figure 4.2, which illustrates them in more detail.

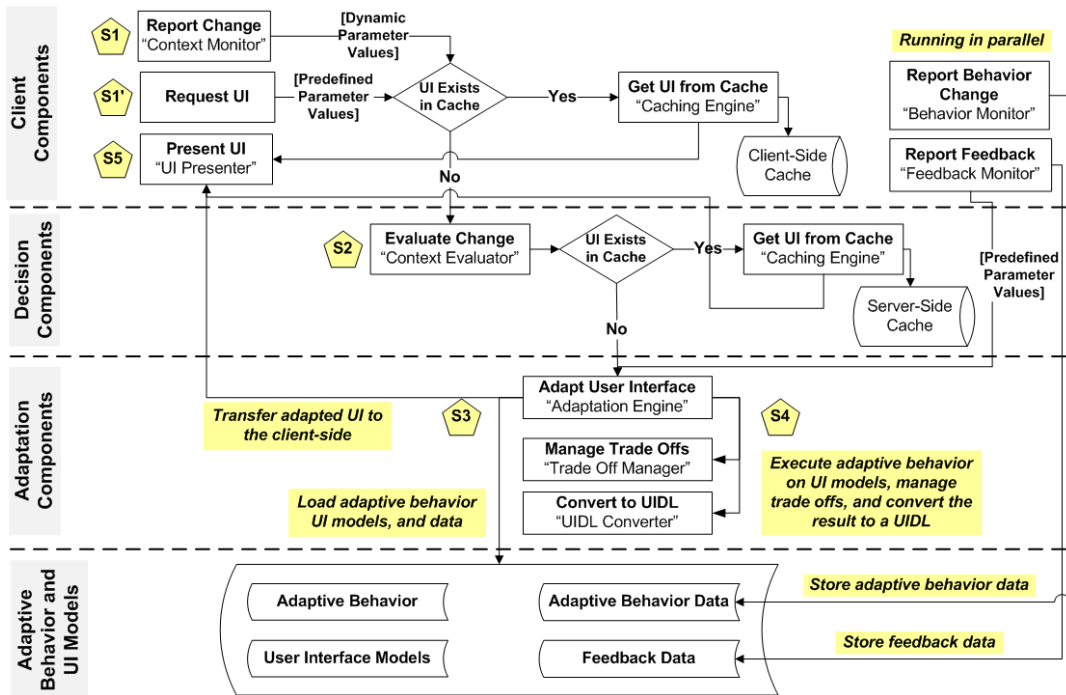


Figure 4.2: User Interface Adaptation Procedure

A direct UI adaptation could occur once a context-change is detected by the *Context Monitor* (S1) and reported to the *Context Evaluator* in case the client-side cache does not have the necessary UI. A decision is made on whether the

UI should be adapted. The server-side cache is checked for an existing version. If the required UI version does not exist in the cache, the adaptation engine is called (**S2**) for obtaining the new UI. The adaptive behavior and UI models are loaded (**S3**) from the database server. The UI is then adapted (**S4**) and sent back to the client-side in order to be presented to the end-user (**S5**). On the other hand, with an indirect adaptation the UI is not adapted while the end-user is working but rather when he or she launches a UI (**S1'**). However, the adaptation procedure goes through the same steps but the UI is adapted based on predefined parameters such as the end-user's computer literacy level rather than dynamic parameters that are detected by the *Context Monitor* such as data collected from sensors.

The *Behavior Change* and *Feedback* monitor(s) constantly run in parallel with the other functionality. Once a behavior change is detected, the new data is stored on the database server. This data is collected over time and processed in order to refine the adaptations. Feedback submitted by the end-users is also stored on the database server in order to refine the adaptations. Since the end-users are manually submitting the feedback, the adaptation engine is called after storing the feedback data in order to directly reflect the change.

4.3 General-Purpose Meta-Model

This section presents and explains our general-purpose meta-model, which offers a high-level design for supporting the development of adaptive UIs based on the Cedar Architecture.

4.3.1 Multi-Aspect Adaptive User Interfaces Meta-Model

The class diagram illustrated in Figure 4.3 presents the part of our meta-model, which supports extensible multi-aspect user interface adaptation with trade-off management capabilities.

Our meta-model links *Adaptation Aspects* (e.g., computer literacy, device, job title, etc.) with *Goals* (e.g., usability, security, etc.). *Goals* could be either crisp or fuzzy and are represented in *Goal Models*.

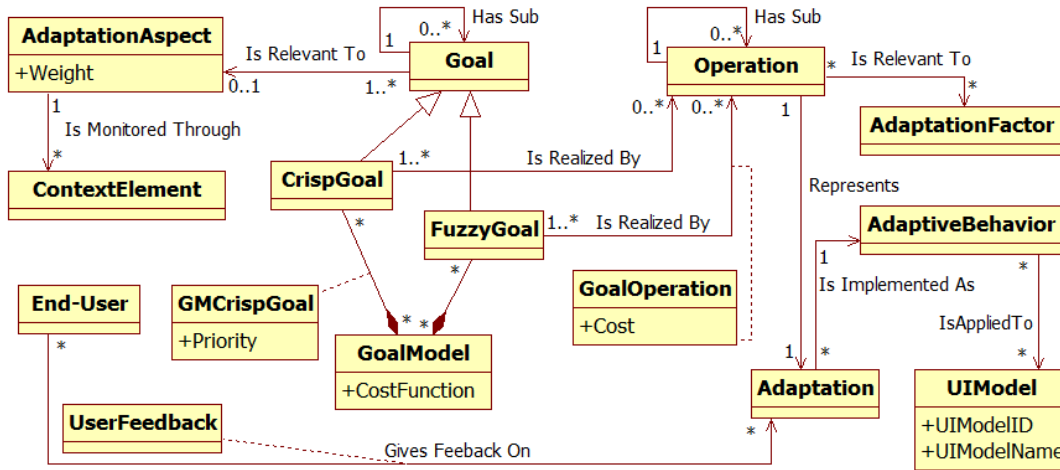


Figure 4.3: Meta-Model for Multi-Aspect Adaptive User Interfaces

Baresi et al. (2010) distinguish between *Crisp Goals*, whose satisfaction is Boolean, and *Fuzzy Goals*, whose satisfaction is represented through fuzzy constraints. The goals represented by our meta-model follow a similar definition. For example, a “device” aspect could be linked to a fuzzy “usability goal”, which dictates how the UI should be adapted to meet the device’s characteristics such as the screen-size. A “job title” aspect could be linked to a crisp “security goal”, which dictates the fields that can be viewed or edited by each user-role. *Crisp Goals* are linked to *Adaptation Aspects* that are realized by *Operations* relevant to Boolean-valued *Adaptation Factors*. An example of an *Adaptation*, which realizes a crisp goal, is a feature-reduction that changes some Boolean properties of UI widgets such as those related to whether the widget is: disabled, read-only, visible, etc. On the other hand, *Fuzzy Goals* are linked to *Adaptation Aspects* that are realized by *Operations* relevant to *Adaptation Factors*, which are represented as numeric sets of values. A layout optimization is a type of *Adaptation* that can realize a fuzzy goal. For example, the type of a selection widget can be changed to: combo-box, radio-buttons, list-box, etc.

To support trade-off analysis, *Crisp Goals* have a *priority* whereas *Fuzzy Goals* define a *cost* for each of their relevant *Operations*. The priority property is used to sort goals for selecting the top one, whereas the cost could be used in a cost function to determine the extent to which each goal can be fulfilled. The change of an *Adaptation Aspect* is captured by monitoring a *Context Element* such as: user-role, environment variable, platform type, etc. When a change

4.3.2 User Interface Levels of Abstraction Meta-Model

```

classDiagram
    class UIModel {
        +UIModelID
        +UIModelName
    }
    class TaskModel {
    }
    class AUIModel {
    }
    class CUIModel {
    }
    class Task {
        +TaskID
        +TaskName
    }
    class AUIElement {
        +AUIElementID
        +AUIElementName
    }
    class CUIElement {
        +Name
    }
    class GraphicalCUIElement {
        +Anchor
        +BackColor
        +Dock
        +IsEnabled
        +FontName
        +FontSize
        +FontStyle
        +ForeColor
        +TextAlignment
        +TabStop
        +IsReadOnly
        +IsVisible
        +LeftPosition
        +TopPosition
        +Height
        +Width
    }
    class TaskRelation {
    }
    class TemporalOperator {
        +Choice
        +Concurrency
        +OrderIndependence
        +Synchronization
        +Disabling
        +SuspendResume
        +Enabling
        +EnablingInfo
    }
    class TaskType {
        +Abstraction
        +Interaction
        +User
        +Application
        +Cooperation
    }
    class AUIElementType {
        +Input
        +Output
        +Container
        +Action
        +Selection
        +Navigation
    }
    class GraphicalCUIElementType {
        +GraphicalCUIElementTypeID
        +GraphicalCUIElementTypeName
        +ToolkitPath
    }
    class GraphicalCUIElementProperties {
        +PropertyName
    }
    class Window {
        +WindowID
    }
    class WinParameters {
        +ParameterName
        +ParameterValue
    }
    class MultiLingualCaption {
        +TextValue
    }
    class Caption {
        +TextID
    }
    class Language {
        +LanguageID
        +LanguageName
    }

    UIModel <|-- TaskModel
    UIModel <|-- AUIModel
    UIModel <|-- CUIModel
    AUIModel <|-- AUIElement
    CUIModel <|-- CUIElement
    CUIElement <|-- GraphicalCUIElement
    GraphicalCUIElement <|-- GraphicalCUIElementType
    GraphicalCUIElement <|-- GraphicalCUIElementProperties

    UIModel "1" -- "*" TaskModel : +Is Embedded In
    UIModel "1" -- "*" AUIModel : +Is Embedded In
    UIModel "1" -- "*" CUIModel : +Is Embedded In
    TaskModel "1" -- "*" Task : +Has
    AUIModel "1" -- "*" AUIElement : +Has
    CUIModel "1" -- "*" CUIElement : +Has
    Task "0..*" -- "0..1" Task : +Depends On
    Task "0..*" -- "0..1" Task : +Has Parent
    Task "1" -- "*" TaskRelation : +Is Related To
    Task "1" -- "*" TaskType : +Has
    AUIElement "1" -- "*" AUIElementType : +Has
    CUIElement "1" -- "*" GraphicalCUIElement : +Has
    GraphicalCUIElement "1" -- "*" GraphicalCUIElementType : +Has
    GraphicalCUIElement "1" -- "*" GraphicalCUIElementProperties : +Has
    CUIModel "*" -- "*" Window : +Is Assigned To
    Window "*" -- "*" WinParameters : +Has
    Window "*" -- "*" MultiLingualCaption : +Has
    MultiLingualCaption "1" -- "*" Caption : +Has
    Caption "*" -- "*" Language : +Has
    Task "1" -- "*" TaskRelation : +Is Related To
    TaskRelation "1" -- "*" TemporalOperator : +Has
    Task "1" -- "*" TaskType : +Has
    AUIElement "1" -- "*" AUIElementType : +Has
    CUIElement "1" -- "*" GraphicalCUIElement : +Has
    GraphicalCUIElement "1" -- "*" GraphicalCUIElementType : +Has
    GraphicalCUIElement "1" -- "*" GraphicalCUIElementProperties : +Has
    CUIModel "*" -- "*" Window : +Is Assigned To
    Window "*" -- "*" WinParameters : +Has
    Window "*" -- "*" MultiLingualCaption : +Has
    MultiLingualCaption "1" -- "*" Caption : +Has
    Caption "*" -- "*" Language : +Has
    Task "1" -- "*" TaskRelation : +Is Related To
    TaskRelation "1" -- "*" TemporalOperator : +Has
    Task "1" -- "*" TaskType : +Has
    AUIElement "1" -- "*" AUIElementType : +Has
    CUIElement "1" -- "*" GraphicalCUIElement : +Has
    GraphicalCUIElement "1" -- "*" GraphicalCUIElementType : +Has
    GraphicalCUIElement "1" -- "*" GraphicalCUIElementProperties : +Has
    CUIModel "*" -- "*" Window : +Is Assigned To
    Window "*" -- "*" WinParameters : +Has
    Window "*" -- "*" MultiLingualCaption : +Has
    MultiLingualCaption "1" -- "*" Caption : +Has
    Caption "*" -- "*" Language : +Has
  
```

This part of the meta-model represents the UI levels of abstraction suggested by the CAMELEON reference framework, namely the task, abstract UI (AUI),

and concrete UI (CUI) models. This part of the meta-model complements the one shown in Figure 4.3.

4.3.2.1 Task Models

We adopted the ConcurTaskTrees (Paterno` 1999) notation for representing the *Task Models*, which are the highest level of abstraction focusing on the activities that the UI is required to support. Following this notation, a *Task Model* is composed of several *Tasks* each having a type from those enumerated by the *Task Type* class. The *Tasks* are connected in a hierarchical manner to indicate parent/child relationships. In addition to these hierarchical relationships, the *Tasks* are connected to each other with logical temporal relationships (*Task Relation*) since there could be interdependency among *Tasks*. For example, a calculated field could depend on values from other fields for calculating its value.

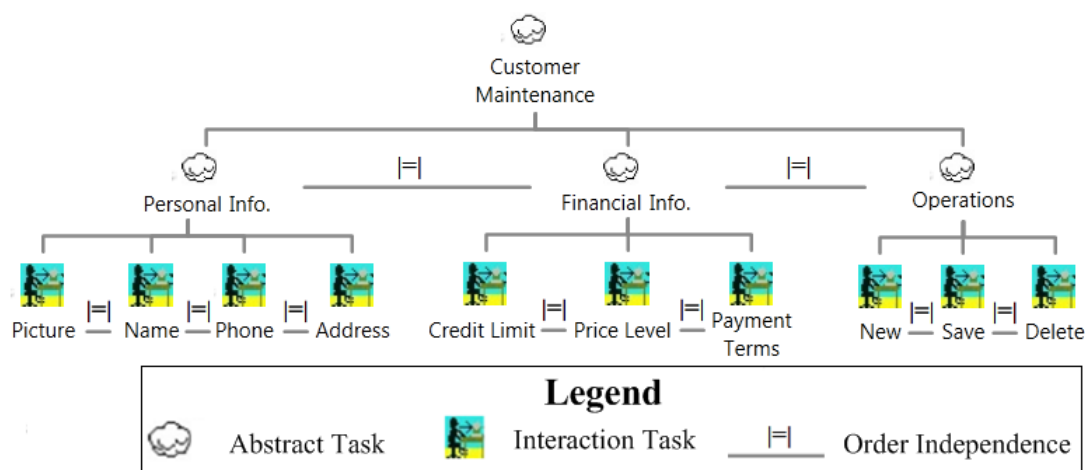


Figure 4.5: Task Model for Customer Maintenance UI

The ConcurTaskTree shown in Figure 4.5 is an example of a task model for a *Customer Maintenance* UI, which is based on the meta-model from Figure 4.4.

4.3.2.2 Abstract User Interface Models

The *AUI Model* is a modality-independent representation of the user interface. It is composed of *AUI Elements* each of which has a type from those enumerated by the *AUI Element Type*. These elements could be grouped inside the model

using *container* elements. Each *AUI Element* could be mapped to many *Tasks* in the *Task Model* and vice-versa.

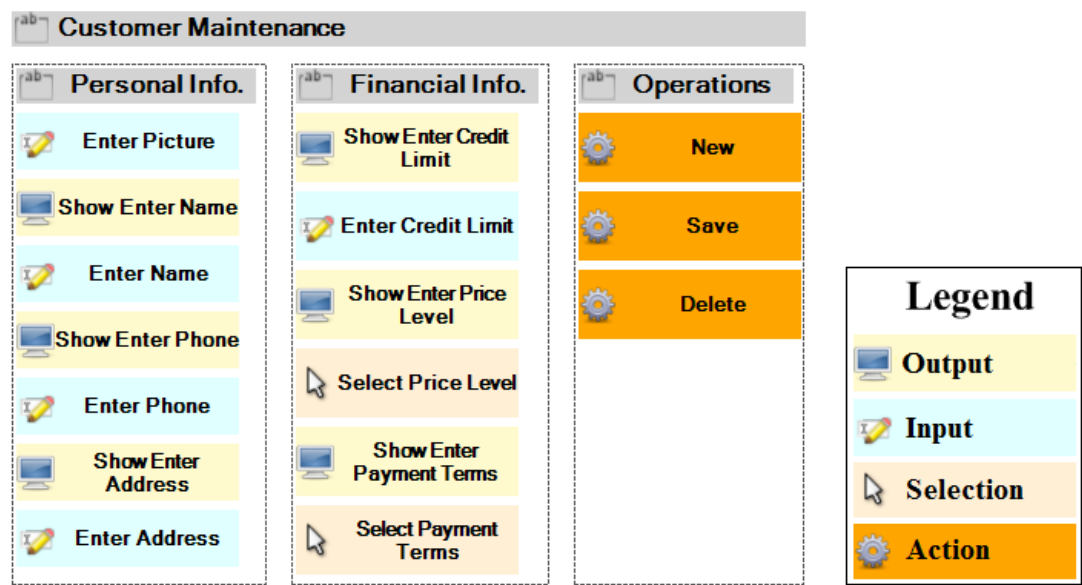


Figure 4.6: Abstract UI Model for Customer Maintenance UI

The *Customer Maintenance* UI is used again as an example in Figure 4.6 to demonstrate an abstract UI model, which was created based on the meta-model from Figure 4.4. The elements on this AUI model are mapped to the tasks of the task model shown in Figure 4.5.

4.3.2.3 Concrete User Interface Models

The *CUI Model* is a modality-dependent representation of the user interface. Therefore, the *CUI Elements* could be of different types each representing a certain modality such as: graphical, character, voice, etc. However, since enterprise applications rely mostly on graphical UIs (GUIs), we only define a *Graphical CUI Element* (widget) class as a specialization of the *CUI Element* class. Nevertheless, the meta-model could be extended in the future for supporting other modalities. The *Graphical CUI Element* has standard properties (e.g., height, width, etc.), which are common for all graphical UI widgets. Additionally, these elements define *Graphical CUI Element Properties*, which

depend on each element's type. For example, a data-grid widget could have a property called “alternating row color”, which is not present in other widgets.

Most enterprise application user interfaces are data entry forms. Therefore, we adopted a relative positioning approach for the layout, whereby *Graphical CUI Elements* could be embedded inside one another (e.g., text-boxes inside a group-box) and positioned using the top and left position properties. Different presentation technologies such: HTML, Java Swing, and Windows Forms, are used in industry for developing enterprise application UIs. These technologies can support the layout approach that we adopted. To support multi-lingual UIs, *Graphical CUI Elements* are assigned a *Multilingual Caption*. Each *Graphical CUI Element* has a type from the *Graphical CUI Element Types* such as: button, combo-box, text-box, etc. These types define a *Toolkit Path* property, which indicates where the UI Presenter (Section 4.2.2.1) component could locate the relevant widget inside the toolkit. For example, if the Windows Forms toolkit is adopted, the *Toolkit Path* property could store the assembly path of the widget (e.g., “System.Windows.Forms.Button”). The end-users can access the UI by activating *Windows*, which are presented as links with user-friendly captions inside a navigation structure such as a menu.

Figure 4.7: Concrete UI Model for Graphical Customer Maintenance UI

The UI shown in Figure 4.7 is the *Customer Maintenance* graphical CUI model, which maps to the AUI model shown in Figure 4.6 and is based on the meta-model illustrated in Figure 4.4.

4.4 Evaluating the Performance of Interpreted Runtime Models

One might think that runtime performance could be a concern with UIs that are interpreted and rendered dynamically. To check the validity of this concern, we ran a preliminary experiment to compare the performance of dynamic UIs to that of UIs represented as compiled code. For this experiment, we chose a *Medical Claims* UI from an existing open-source dental practice management system called OpenDental⁹. Although this software application is not as large-scale as ERP systems, it was enough for this preliminary study because it uses WIMP-style UIs that are common in enterprise applications, and it adopts a compiled presentation technology namely, Windows Forms. A large-scale enterprise application was used in the comprehensive evaluation discussed in Chapter 7. The selected *Medical Claims* UI has 87 widgets of 9 different types. We reverse-engineered it from code into a CUI model representation conforming to the meta-model in Figure 4.4.

We tested the performance of the dynamic UI versus the code-based one. The dynamic UI had to load all the widgets at runtime from a database, whereas the code-based one is a compiled edition of which an instance can be created and displayed on the screen. This test was conducted on a single machine running Windows 7 with an Intel Core 2 Duo 2.93 GHz CPU and 4 GB of RAM. Here, we should note that the initial code-based UI was developed in Windows Forms, whereas the dynamic one uses the Windows Presentation Foundation (WPF). We meant to use a different presentation technology in order to show the portability of our approach.

Both the code-based and interpreted model-driven versions of the *Claims* UI were sequentially loaded and closed 1000 times. The time, in milliseconds (ms), was plotted on the graph illustrated in Figure 4.8.

The mean loading-time was 16.1 ms for the code-based UI and 12.46 ms for the interpreted runtime model-driven one. We can notice that the dynamic UI took slightly more time when it was loaded the first time (25 ms) then the effect

⁹ www.opendental.com

of the caching allowed a significant drop in the time. We noticed the phenomenon of having a slightly faster (< 5 ms) loading time with dynamic UI. We attribute this variation to the different rendering technology. Overall, we can say that utilizing our dynamic approach does not incur any negative impact on performance.

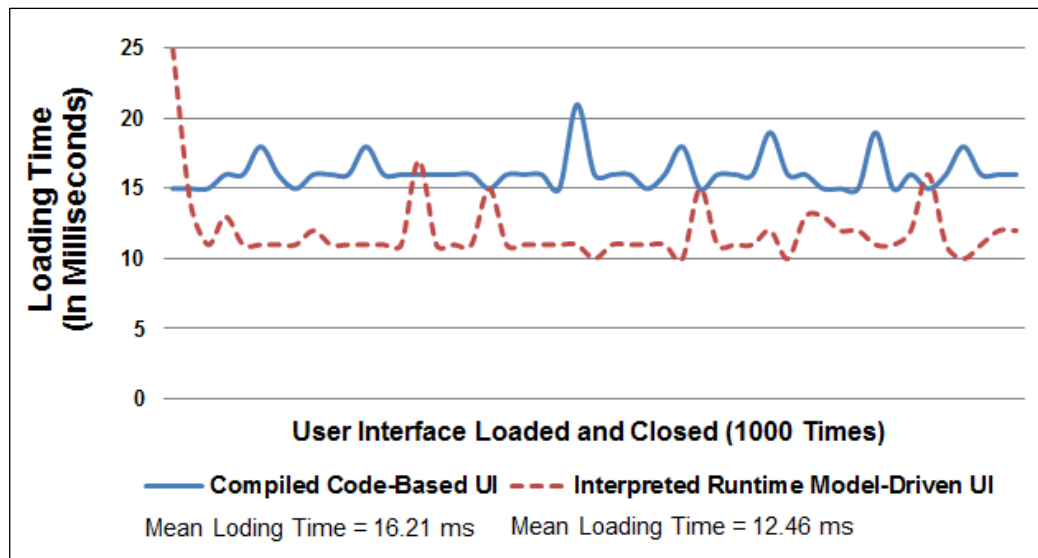


Figure 4.8: Performance Comparison between a UI based on Interpreted Runtime Models and a Code-Based One

We did not perform any adaptation in this experiment in order to maintain the same number of widgets when comparing the two user interfaces.

4.5 Chapter Summary

This chapter presented the Cedar Architecture for answering research question **Q1**, which we established in Chapter 3. This architecture serves a reference for stakeholders interested in developing adaptive model-driven user interfaces for enterprise applications. It promotes the use of interpreted runtime models and offers components for filling the gaps that we identified in Chapter 2, such as: trade-off management, user-feedback, etc.

We provided a general explanation of the layers and components comprising the Cedar Architecture. We also explained the steps it presents for adapting a UI based on the context-of-use, either directly while the user is working, or

indirectly by proposing the adapted UI as an alternative. Furthermore, we presented a general-purpose meta-model, which offers a high-level design for supporting the development of adaptive model-driven UIs based on the Cedar Architecture. The meta-model supports model-driven UI representation on multiple levels of abstraction and multi-aspect adaptive behavior with trade-off management capabilities for conflicting aspects.

Finally, we conducted an experiment, which showed that by employing data caching, UIs that are based on interpreted runtime models can load as efficiently as those that are based on compiled code. Therefore, runtime performance is not a point of concern when using interpreted runtime models for developing user interfaces.

In the next chapter, we present Role-Based User Interface Simplification (RBUIS), an adaptation mechanism based on the Cedar Architecture. RBUIS offers a technical realization for various high-level components of the Cedar Architecture. It can be used for providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. We also presented a trade-off analysis technique, which complements RBUIS and realizes the trade-off management component of the Cedar Architecture.

5

RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior

“Everything should be made as simple as possible, but not simpler.”

— Albert Einstein

This chapter presents the Role-Based UI Simplification (RBUIS) mechanism, which is an adaptation method based on the Cedar Architecture. RBUIS can be used for simplifying enterprise application UIs through engineering role-based adaptive behavior. We define UI simplification as a mechanism for improving usability through adaptive behavior by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. RBUIS supports UI simplification for an extensible number of adaptation aspects. Therefore, trade-off analysis can help in managing conflicting adaptation choices when multiple aspects simultaneously impact the same UI factors. Hence, this chapter also presents an aspect-level trade-off analysis mechanism, which uses goal models, Pareto optimality, and cost functions, complementing RBUIS.

5.1 Introduction

The functionality of software applications tends to increase with every release thereby increasing the visual complexity. This phenomenon, referred to as “bloatware” (McGrenere 2000), has a negative impact on usability especially for end-users who do not require the complete UI feature-set. Also, end-users can have different layout preferences. Both feature-set and layout related choices, can be affected by several aspects such as: skills (Uflacker & Busse

2007), culture (Reinecke & Bernstein 2011), etc. Although several approaches have been proposed for adapting UIs to various contexts-of-use, little work has focused on simplifying enterprise application UIs through engineering adaptive behavior. This chapter presents Role-Based UI Simplification (RBUIS), a mechanism for improving usability by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. We define a feature as a functionality of the software system and a minimal feature-set as the UI sub-set with the least features required by an end-user to perform a job. An optimal layout is the one that maximizes the satisfaction of the constraints imposed by a set of aspects. An optimal layout is achieved by adapting concrete widget properties such as: type, grouping, size, location, etc. RBUIS is based on the Cedar Architecture, which we presented in Chapter 4, and implements several of its high-level components.

Since multiple aspects can simultaneously impact the same UI factors, trade-off analysis is vital for managing conflicting adaptation choices. Supple (Gajos et al. 2010), for example, elicits a cost function for each end-user to select the optimal UI factors (e.g., widget type). However, its trade-off analysis technique only supports a fixed and limited number of adaptation aspects, namely vision and motor impairment. Even though cost functions are essential for selecting the optimal UI factors for each end-user, in some scenarios the end-users might prefer certain aspects over others such as: mobility over detail or vision over screen-size, depending on the changing context-of-use. Therefore, this chapter also presents an aspect-level trade-off analysis technique, which complements RBUIS, and supports an extensible number of adaptation aspects. This technique can be parameterized with aspectual weights that are supplied by the end-users through a feedback mechanism.

The contributions presented in this chapter, answer the following research questions, which were established in Chapter 3:

Q2: What UI adaptation technique, based on the proposed reference architecture (Cedar Architecture), can minimize the feature-set and optimize the layout of enterprise application UIs according to the context-of-use for an extensible number of adaptation aspects and factors, and support the addition of both visual and code-based adaptive behavior as needed?

Q3: What trade-off management mechanism can support multi-aspect trade-off analysis that works with the devised UI adaptation technique (Q2)?

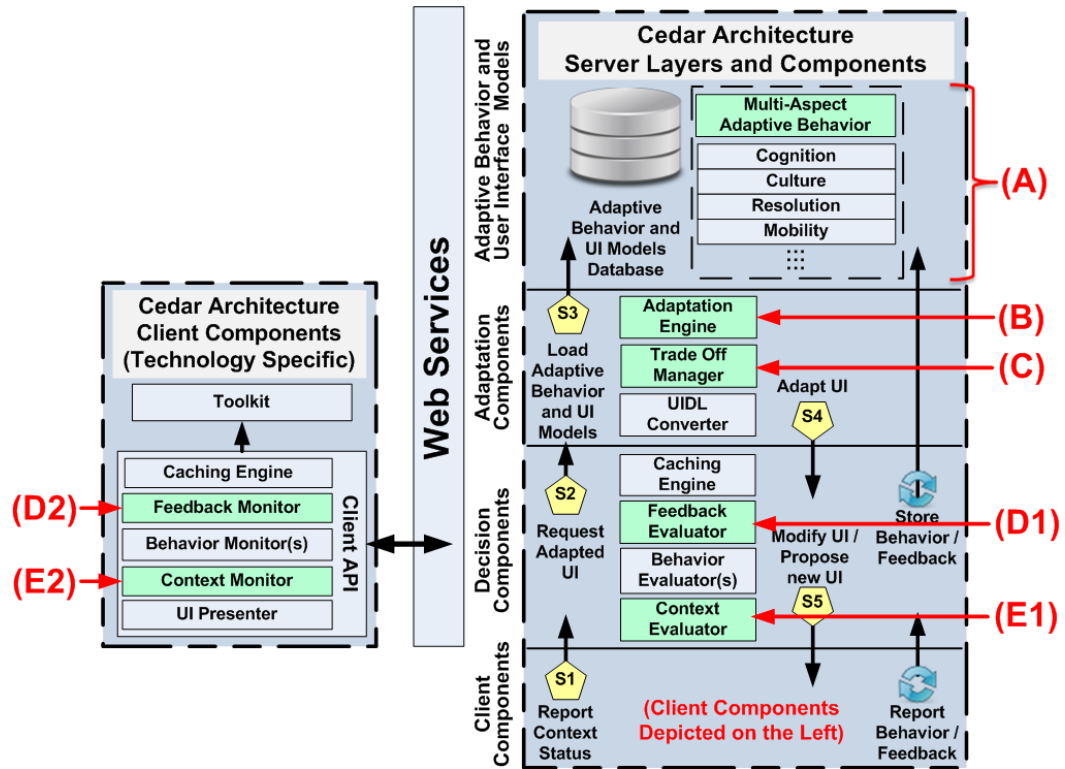


Figure 5.1: Cedar Architecture Components that are realized by RBUIS

The parts of the Cedar Architecture (Chapter 4), which are labeled with letters A to E in Figure 5.1, are realized by RBUIS as will be described in this chapter. The database in the *Adaptive Behavior and UI Models* layer (A), stores the UI models: task, AUI, and CUI. In RBUIS, this layer also stores role-task assignments and RBUIS rules for feature-set minimization (Section 5.3), and adaptive behavior workflows and scripts for layout optimization (Section 5.4). The *Adaptation Engine* (B) minimizes the feature-set according to the task-role assignments that were done on the task model (Section 5.3.3) and optimizes the layout by executing the adaptive behavior workflows on the CUI model (Section 5.4.2). The *Trade Off Manager* (C) is realized by the trade-off analysis technique presented in Section 5.6. The Feedback Evaluator (D1) and Feedback Monitor (D2) are realized by RBUIS's feedback mechanism, which is presented in Section 5.5. In RBUIS, the *Context Evaluator* (E1) and *Context Monitor* (E2) components simply check the roles of the logged-in user in order to offer the

relevant adaptations. The *UIDL Converter*, *Caching Engine* (client and server), and *UI Presenter* components were realized in Chapter 4 to carry out the performance evaluation that was presented in Section 4.4. The adaptation procedure followed by RBUIS is as described in Section 4.2.4.

5.2 Role-Based User Interface Simplification (RBUIS)

To simplify UIs, we need to provide the end-users with a minimal feature-set and an optimal layout based on the context-of-use. In the case of the feature-set, the initial UI design contains all the features hence it is without constraints. Yet, initial designer layout related choices such as: widget types and grouping have to be the least constrained, for example in terms of the screen-size. The designers devise the UI for the least constrained profile at design-time. Afterwards, a role-based approach is used to simplify it at runtime based on the context-of-use. Role-based modeling has been used for adapting the components of software applications (Piechnick et al. 2012). However, our approach is oriented towards merging access control with model-driven UIs to achieve UI simplification.

The standard for role-based access control (RBAC) can be used by enterprises for protecting their digital resources (Ferraiolo et al. 2001). In RBAC, “users” are assigned “roles”, which are in turn assigned permissions on “resources”. In our case, the users are the enterprise employees logging into the system with their accounts, and the resources that we want to apply roles to, are the UI models and the adaptive behavior. We merged the role-based approach with UI simplification to create Role-Based User Interface Simplification (RBUIS), in the spirit of RBAC. In RBUIS, roles are divided into groups representing the adaptation aspects based on which the UI will be simplified. RBUIS is applied after deploying the software in the enterprise. Managing this process could be a joint cooperation between personnel from the software company in charge of the deployment process and the enterprise’s IT personnel.

RBUIS comprises the following elements, which support feature-set minimization and layout optimization:

- **Role-Based UI Models** support feature-set minimization through role assignment to task models for providing a minimal feature-set based on the

context-of-use. This approach allows a practical realization of the concept of multi-layer interface design (Shneiderman 2003).

- **Role-Based Adaptive Behavior** supports layout optimization through role assignment to workflows that represent adaptive UI behavior visually and through code. The workflows are executed on the concrete user interface (CUI) models in order to optimize the layout. The workflow-role assignment specifies which workflows to execute for each group of end-users.
- **User Feedback** is supported for refining the adaptations if necessary, by allowing the end-users to reverse feature-set minimizations and layout optimizations, and to choose alternative layout optimizations when possible.

5.3 Minimizing the User Interface's Feature-Set

The meta-model shown in Figure 5.2 depicts how RBUIS is applied to the task model. The classes that have a white background color already appeared in the general purpose meta-model illustrated by Figure 4.3 and Figure 4.4 in Chapter 4.

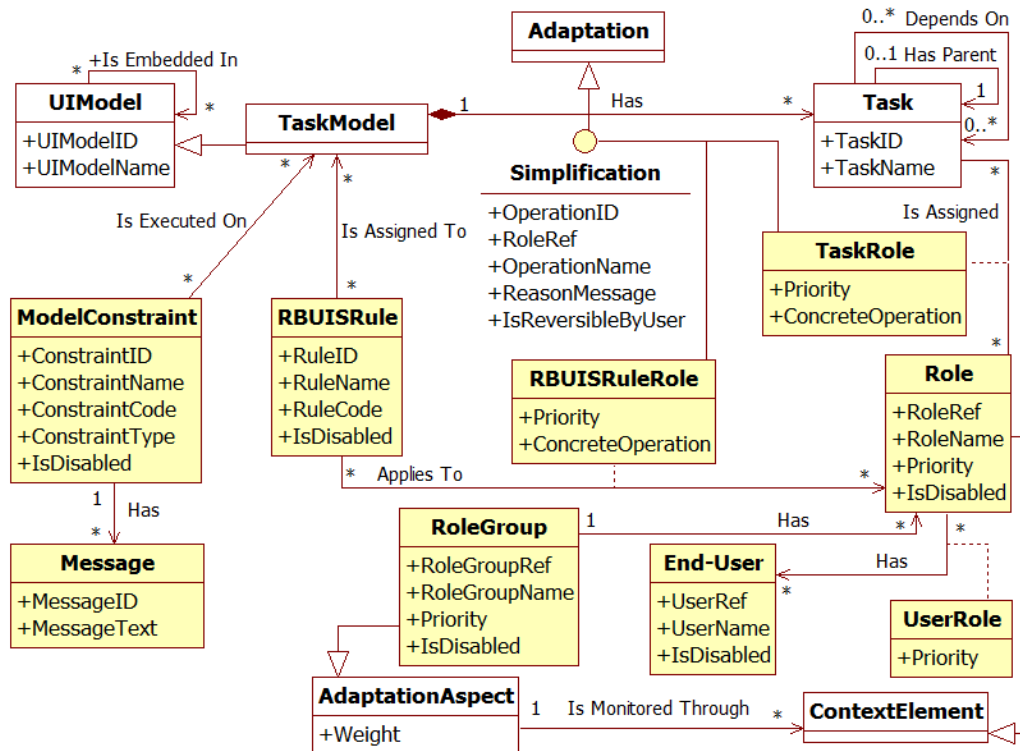


Figure 5.2: Meta-Model of Applying RBUIS to the Task Model

We adopted the concept of multi-layer interface design for minimizing the feature-set. This concept allows end-users to control different sub-sets of the UI at any moment. For example, novice users could be given access to layer 1 and as their expertise develops, they could gain access to the upper layers at any time. RBUIS provides a practical approach for controlling the different UI layers.

5.3.1 Feature-Set Minimization with RBUIS

Applying RBUIS to task models can minimize the feature-set by revoking access to *Tasks* based on *Roles* hence achieving a role-based multi-layer interface design. *Roles* such as: novice/expert and cashier/accountant are allocated to *Role Groups* such as: computer literacy level, job title, etc. Since we are initially designing the UI for the least constrained profile, the default policy grants all the *Roles* access to all the *Tasks*. This could be considered as a layer containing all the features. Afterwards, access could be revoked by allocating roles to tasks, thereby creating separate layers to which end-users could gain access based on their roles. Since the *End-Users* can be allocated multiple *Roles* from the existing *Role Groups*, priorities are used to provide enough flexibility for specifying how roles override each other. Upon assigning the access rights to block tasks based on roles, a property we call “*concrete operation*” can be used to specify whether to make a task invisible, disable it, or fade it until first use. The task model is mapped to the AUI model, which is in turn mapped to the CUI model to hide, disable, or fade the relevant UI widgets.

As indicated in Chapter 4 (Section 4.3.2.1), we chose the ConcurTaskTrees (Paterno 1999) notation for representing the task models. One advantage of using this notation with RBUIS is its support of temporal constraints on the task relationships. Bergh et al. (2010) indicated that these constraints can help in determining the dependencies among tasks. To determine if simplifying a task affects other tasks, we present the algorithm in Section A.3 of Appendix A.

5.3.2 Less Time Consuming Access Rights Allocation

Enterprise applications can encompass a large number of tasks, which are used by hundreds of users. Therefore, we need to make the allocation of access

rights on the task models consume as little time as possible. Traditionally, enterprise application users are allocated roles. This practice can be considered as a positive starting point. We resort to the following points to minimize the time taken to allocate roles to tasks in the task models:

- A default policy grants all the roles access on all the application’s task models, hence making it only necessary to override this policy where access should be revoked. Each task is implicitly allocated a fixed role called “All-Roles”, which represents all the roles in the system and is granted access to execute the task. Access to the task is revoked to all other explicitly assigned roles.
- Sub-tasks inherit the access rights of the parent tasks while maintaining the ability to override these rights.
- In some cases, the same functionality is replicated in many places within the application. Usually developers create visual components (CUI level), which can be reused in different places. By making task models reusable within one another, access rights allocated to a task model can roam with it whenever it is used again, while maintaining the ability to override the initial rights. This feature is illustrated in Figure 5.2 with the recursive relationship “Is Embedded In” on the *UI Model* class.
- Rules can be defined and applied to multiple task models based on each task’s properties such as: identifier, name, type, etc. *RBUIS Rules* (Figure 5.2) are defined in the form of conditions using SQL syntax. *RBUIS Rules* are assigned the *Task Models* on which to execute, and the *Roles* for which they apply. One basic example is revoking access of the role *Cashier* on all *interaction* tasks containing the phrase “enter discount” in the task name.

5.3.3 Applying RBUIS to Task Models at Runtime

Based on the Cedar Architecture (Figure 4.1), the UI models are loaded on the server and the adaptation engine applies RBUIS at runtime. To apply concrete operations such as hide and disable on the CUI, the task model is mapped to the AUI, which is in turn mapped to the CUI. A certain order should be followed to perform the elimination since each end-user can be allocated multiple roles

simultaneously. The meta-model, shown in Figure 5.2, allows the assignment of priorities on different levels including: *Role Group*, *Role*, *Task Role*, and *User Role*. Task-based assignments have a higher priority than rule-based ones unless specified otherwise. The following example demonstrates the process of managing priorities, assuming that they were set at the *Task Role* level:

- **UserA:** Novice, Manager
- **TaskX:** 1. All-Roles (Allow) 2. Accountant (Deny-Hide)
 3. Novice (Deny-Disable)

An excerpt of our algorithm is shown in Listing 5.1. The full version is included in Section A.1 of Appendix A. Following this algorithm, *UserA* is allowed to perform *TaskX* since the *Manager* role has the highest priority. In contrast, if the *Novice* role had a higher priority than *All-Roles*, then *UserA* would have been denied access to *TaskX*, hence disabling its CUI as indicated by the concrete operation.

Listing 5.1: Feature-Set Minimization (Excerpt)

```

1.  Simplify-Task (TaskID, UserRoles[], TaskRoles[], UIModel)
2.  foreach ur in UserRoles // Determine the Primary Role
3.      tr ← TaskRoles.GetRole(ur.RoleRef)
4.      if tr = null then tr ← TaskRoles.GetRole(All-Roles)
5.      ur.Priority ← tr.Priority;
6.  UserRoles.OrderBy(Priority)
7.  PrimaryRole ← UserRoles.First()
8.  if PrimaryRole.RoleRef ≠ All-Roles
9.      // Apply Concrete Operation to CUI
10.   blkdAUI ← GetBlkdAUI(TaskID, UIModel.TMToAUIMap)
11.   blkdCUI ← GetBlkdCUI(blkdAUI, UIModel.UIToCUIMap, UIModel.CUI)
12.   foreach element in blkdCUI
13.       switch PrimaryRole.ConcreteOperation
14.           case Hide: element.Visible ← false; break;
15.           case Disable: element.ReadOnly ← true; break;
16.           case Protect: element.ReadOnly ← true;
17.                       element.MaskChar ← '*'; break;
18.           case Fade: element.Opacity ← '30%'; break;

```

The running time of our algorithm is established to be polynomial: $O(m \times (n \times l \times p \times (2 \times j \times \log j + k) + n))$, where m = number of task models, n = num. of tasks in a task model, j = num. of user-roles, k = num. of blocked CUI elements for a task, p = num. of parent tasks for a task, and l = num. of task-roles.

5.3.4 Model Checking Using SQL

Since the access rights are being allocated by humans, model checking is needed to ensure that critical constraints are not violated. This type of checking allows our support tool to issue appropriate warnings and errors. Several techniques exist for defining and evaluating constraints on models. For example, the Object Constraint Language (OCL) can be used to define constraints on UML diagrams. There are also several technologies that have been used for model checking such as: Z3 (z3.codeplex.com), Formula (bit.ly/MSTFormula), etc. In our case, we need to define constraints on task models represented by CTTs. Since our approach is based on the Cedar Architecture, all the models are stored in a relational database. Hence, model checking can be performed using SQL, which is more familiar to many software developers and IT personnel than constraint languages like OCL. The meta-model in Figure 5.2 supports model-checking using *Model Constraints*, which are executed on *Task Models*. Following, is an example of a constraint to which an SQL-based solution is provided in Listing 5.2.

Constraint: A sub-task should not be blocked for all the roles currently assigned to end-users because it will not be accessible by any end-user in the system.

Listing 5.2: Task Model Constraint Example using SQL

```
With SelTasks as (Select TM.TaskModelID, TM.TaskModelName, TK.TaskID,
TK.TaskName From TaskModel as TM Inner Join TaskModelTask as TK On
TM.TaskModelID = TK.TaskModelID Where TaskModelID in (@ModelIDs)),
UserAccessOnTasks as (Select TaskModelID,TaskID, COUNT(case
UR.CanExecuteTask when 1 then 1 else null end) as CanExecuteCount From
SelTasks Cross Apply LoadSortedUserRoles(TaskModelID,TaskID) as UR Where
UR.UserRolePriority = 1 Group By TaskModelID,TaskID)
Select SelTasks.* From SelTasks ST Inner Join UserAccessOnTasks UAT
On ST.TaskID = UAT.TaskID and ST.TaskModelID = UAT.TaskModelID Where
CanExecuteCount= 0
```


Constraints are associated with task models through a system variable (“@ModelIDs”). Predefined SQL functions such as “LoadSortedUserRoles” can be used in model constraints and extended when necessary. In this case, the function loads the user-identifiers and their assigned roles, and sorts them by the priority of execution according to a certain task. The SQL statement returns the tasks that are violating the constraint.

5.3.5 Feature-Set Minimization Example

The example shown in Figure 5.3 is part of a task model, which represents a *Customer Maintenance* UI that is common in ERP systems.

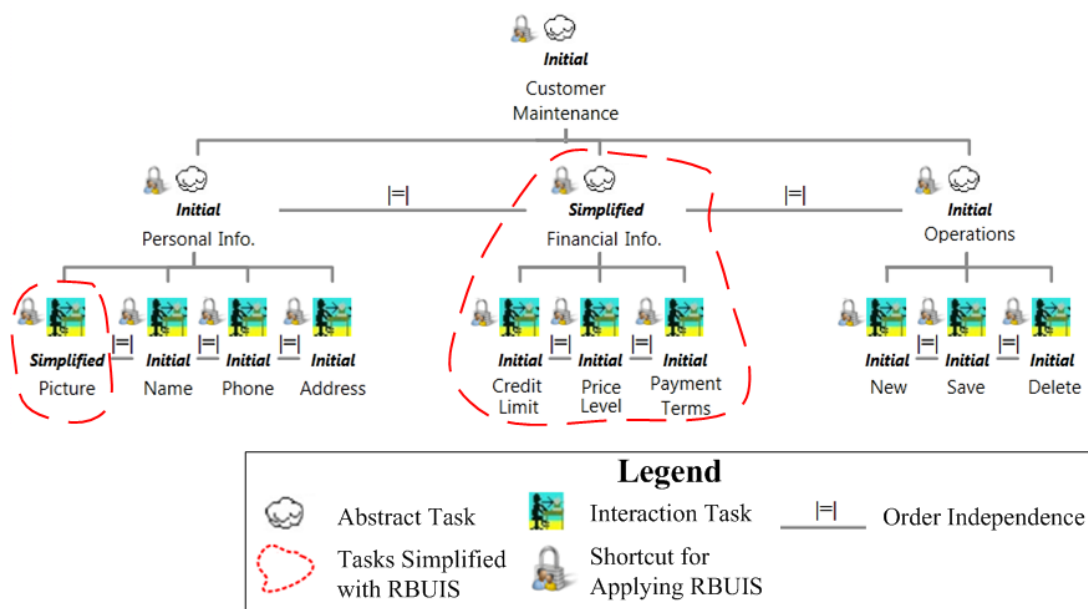
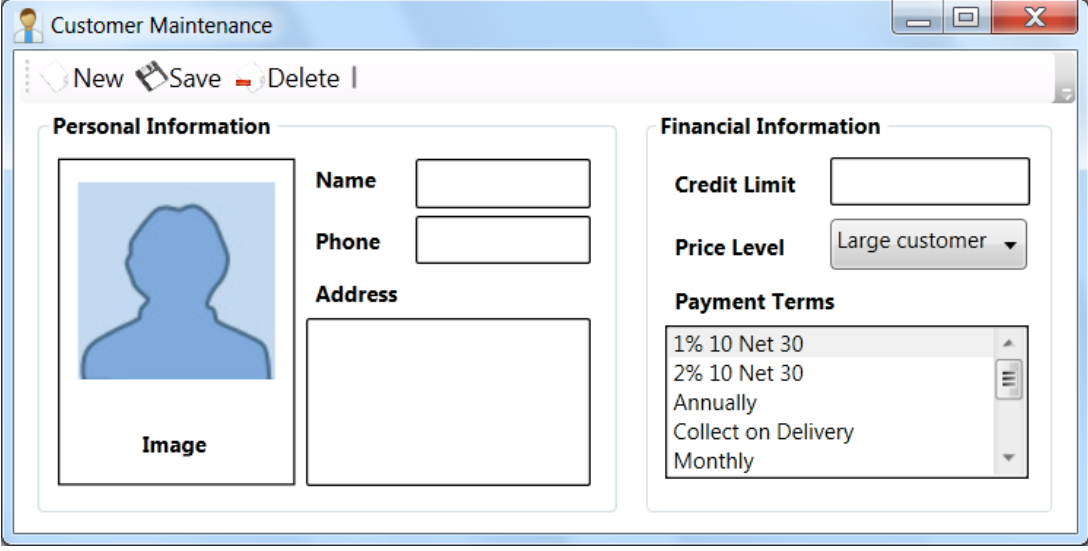


Figure 5.3: Simplified Customer Maintenance Task Model

The lock-shaped buttons allow RBUIS to be applied on any task. In this case, the tasks called *Financial Information* and *Picture* (encircled by a dashed line) are marked as simplified, indicating that RBUIS has been applied. In the case of the *Financial Information* task, the access rights will get inherited by its sub-tasks. We considered a role called *Cashier* requiring a version of the UI showing only the *Name*, *Phone*, and *Address* fields. This role allows end-users working as cashiers, to enter the initial information for a new customer on the counter, without having to handle additional details that can be added later. The initial

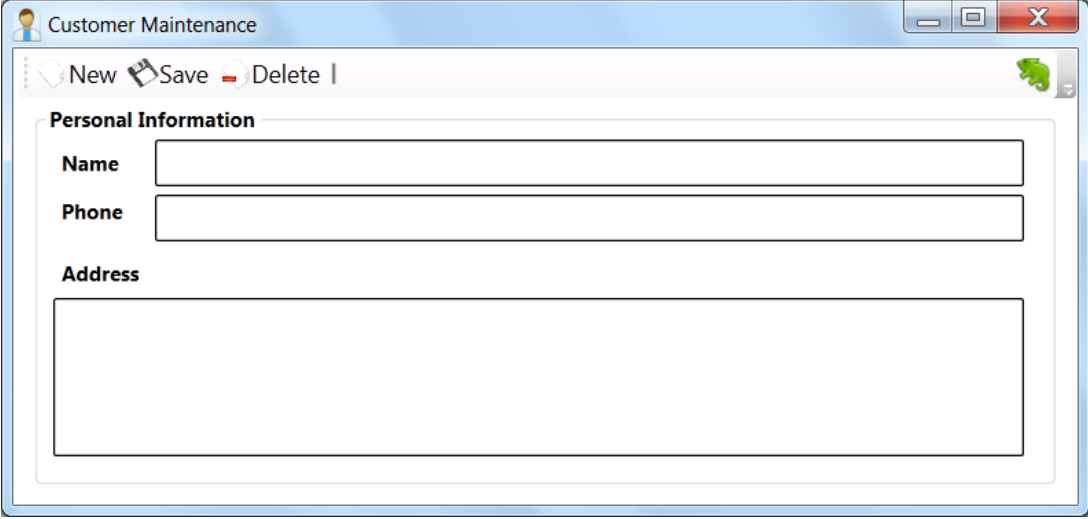
version of the FUI and the one simplified for the role *Cashier* are illustrated in Figure 5.4a, and Figure 5.4b respectively. In this example, the concrete operation in RBUIS was set to “Hide”, hence the widgets became invisible.

(a) Initial Fully-Featured UI Version



The screenshot shows a window titled "Customer Maintenance" with a standard Windows-style title bar (minimize, maximize, close buttons). Below the title bar is a toolbar with icons for "New", "Save", and "Delete". The main content area is divided into two panels. The left panel, titled "Personal Information", contains a placeholder for a customer image (labeled "Image") and input fields for "Name", "Phone", and "Address". The right panel, titled "Financial Information", contains input fields for "Credit Limit" and "Price Level" (which is a dropdown menu currently showing "Large customer"), and a list box for "Payment Terms" with options: "1% 10 Net 30", "2% 10 Net 30", "Annually", "Collect on Delivery", and "Monthly".

(b) Minimized Feature-Set UI Version for Role “Cashier”



The screenshot shows the same "Customer Maintenance" window, but with a minimized feature set for the "Cashier" role. The "Financial Information" panel is completely hidden. The "Personal Information" panel remains, showing the "Name", "Phone", and "Address" input fields. The toolbar and title bar are still present.

Figure 5.4: Feature-Set Minimization of Customer Maintenance UI

5.4.1 Layout Optimization with RBUIS and Workflows

The representation of adaptive behavior has a great impact on the extensibility of any adaptive system. Many adaptive UI state-of-the-art systems employ an arbitrary design that hardcodes adaptation behavior within the software application, thereby severely minimizing its reusability and extensibility. A graphical tool was suggested for hiding the complexity of defining UI adaptation rules (López-Jaquero et al. 2009). However, this tool might not be able to handle all possible adaptation scenarios due to its limited use of a high-level visual representation.

To balance between ease-of-use and flexibility, our approach combines high-level adaptation operations and low level programming constructs by using both visual and code-based representations. Workflows are not strange to enterprise applications due to their use for devising customizable and reusable business rules that can be separated from the software application's code. With appropriate tool support, workflows can also provide visual programming constructs such as: control structures, error handling, etc. It is also possible to define code-based adaptation operations, which can integrate within the visual workflows.

As depicted by the meta-model in Figure 5.5, our approach uses *Adaptive Behavior Workflows*, which can encompass *Adaptive Operations* implemented using both: (1) *Visual Programming Constructs*, and (2) *Compiled Code Libraries* and *Dynamic Scripts*. The workflows are executed at runtime on the *CUI Models* to perform the necessary adaptation.

To implement the workflows in practice, we are using the Windows Workflow Foundation (WF), which is part of the .NET Framework. WF provides the ability to visually design activity workflows using a rich set of constructs, which can be saved in an XML-based format. The XML can be reloaded and executed when an adaptation is needed. Furthermore, the supported constructs can be extended through external compiled class libraries developed in C# or VB.NET. We have used this capability to develop a construct that can be embedded in a workflow for executing non-compiled script code. We currently support Iron Python but it is possible to add other scripting or transformation languages (e.g., XSLT) in the future.

5.4.2 Applying RBUIS at Runtime using Workflows

Layout optimization is also based on the Cedar Architecture (Figure 4.1). After the feature-set is minimized, the adaptation engine executes the workflows on the CUI. Afterwards, the FUI is transferred to the client machine to be presented to the end-user. The process of selecting the workflows to be applied on the CUI based on the end-user's role is shown in Listing 5.3, through an excerpt of our algorithm. The full version is included in Section A.2 of Appendix A. In this excerpt, we assume that the priority is read from the *Role* class (Figure 5.5). The running time of our algorithm is established to be polynomial: $O(2 \times m \times \log m + 2 \times n \times \log n)$, where m = number of user-roles and n = number of workflows to be executed. This algorithm selects the workflows to be applies merely based on a priority. A more advanced trade-off management mechanism will be explained in the coming sections.

Listing 5.3: Layout Optimization (Excerpt)

```

1. Optimize-Layout (UserRoles[], Roles[], UIModel, LayoutID)
2.   foreach ur in UserRoles // Determine the Primary Role
3.     tr ← Roles.GetRole(ur.RoleRef)
4.     if tr = null then tr ← Roles.GetRole(All-Roles)
5.     ur.Priority ← tr.Priority;
6.   UserRoles.OrderBy(Priority)
7.   PrimaryRole ← UserRoles.First()
8.   WorkflowsToExecute[] ← GetWorkflows(PrimaryRole, LayoutID)
9.   WorkflowsToExecute.OrderBy(ExecutionOrder)
10.  foreach workflow in WorkflowsToExecute // Execute Workflows
11.    workflow.Execute(UIModel) // Execution Time Depends on Content

```

5.4.3 Layout Optimization Example

This example builds on the one presented in Section 5.3.5. We consider two roles: *Sales Officer* and *Novice*. A *Sales Officer* end-user requires the fully-featured UI illustrated in Figure 5.4a. On the other hand, a *Novice* end-user requires layout optimizations, which make functions accessible through on-

screen buttons rather than a context-menu, and trading list-boxes for radio-buttons to fit more items on the screen.

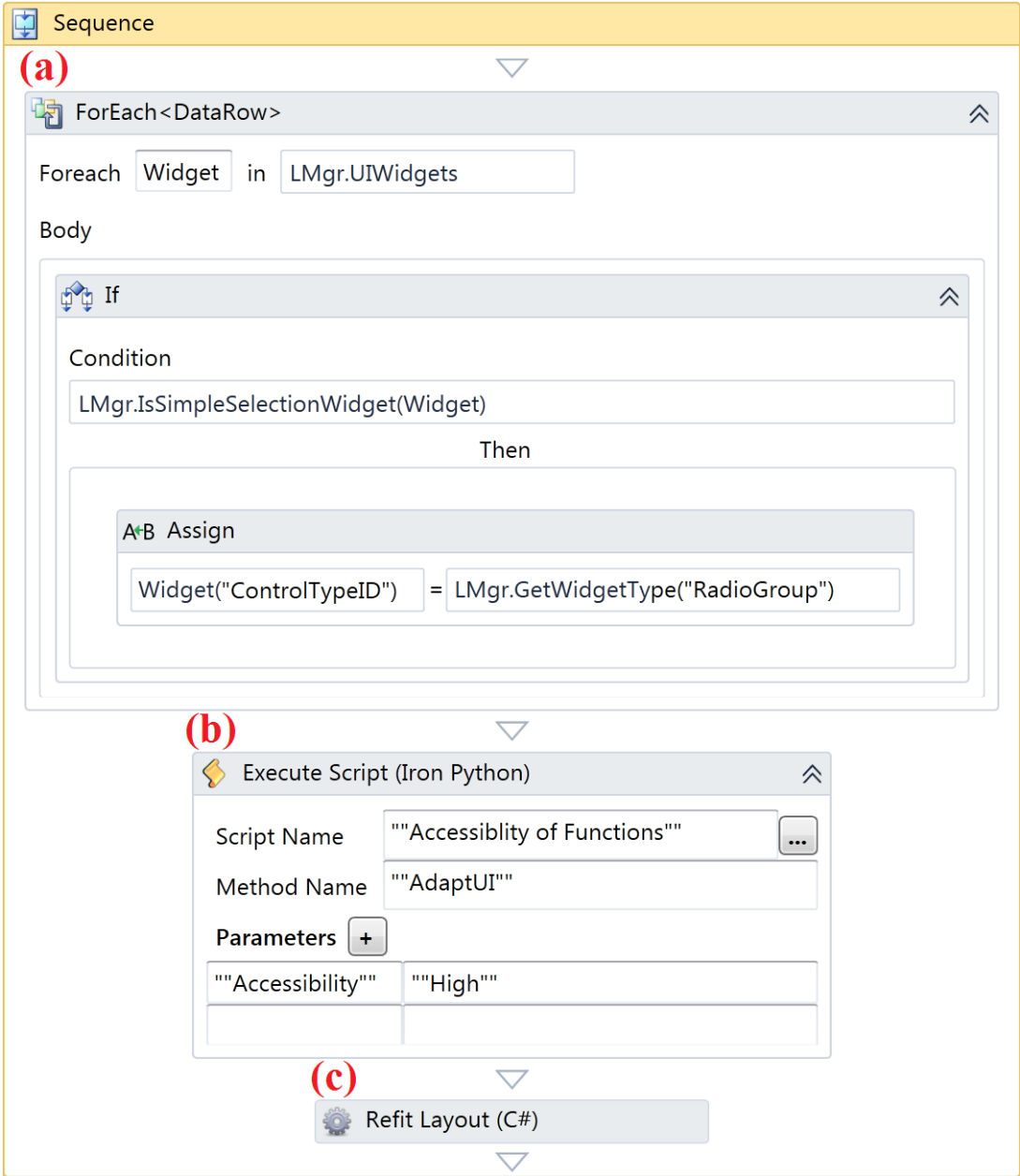


Figure 5.6: Layout Optimization Adaptive Behavior Workflow

The workflow illustrated in Figure 5.6, represents the adaptive behavior by using the following three techniques:

- Visual programming constructs are used to substitute list-boxes with groups of radio-buttons as shown in Figure 5.6a.

A “ForEach” loop is used to iterate around the UI widgets, which are represented in a CUI model and accessed through a helper class called *LMgr* (layout manager). An “If” condition checks whether a CUI element is a simple selection widget (e.g., combo-box, list-box, etc.), in order to change its *Control Type ID* property to *Radio Group*.

— An Iron Python script is called to set the accessibility of functions to *High* as shown in Figure 5.6b.

The script name is selected in addition to the name of the method to be triggered, which in this case is: *Adapt UI*. This method is passed a value of *High* to its *Accessibility* parameter. The workflow internally triggers the Iron Python script shown in Listing 5.4.

Listing 5.4: Iron Python Script for Changing the Accessibility of Functions

```
1. import sys
2. def AdaptUI(UIModelMgr, Accessibility):
3.     UIModelMgr.SetPropertyValue("AccessibilityOfFunctions",
        Accessibility)
```

— A C# algorithm is called for refitting the UI layout as shown in Figure 5.6c.

The screenshot shows a window titled "Customer Maintenance" with a standard Windows-style title bar (minimize, maximize, close buttons). Below the title bar is a toolbar with icons for "New", "Save", and "Delete". The main content area is divided into two primary sections: "Personal Information" on the left and "Financial Information" on the right.

Personal Information Section:

- Contains a placeholder image of a person's silhouette.
- Below the image are three icons: a green plus sign, a red minus sign, and a green document icon, with the label "Image" below them.
- Form fields for "Name", "Phone", and "Address". The "Address" field contains the text "B I U T".

Financial Information Section:

- Form fields for "Credit Limit" and "Price Level". The "Price Level" dropdown menu is currently set to "Large customer".
- A section titled "Payment Terms" containing a group of radio buttons for selection:
 - 1% 10 Net 30
 - 2% 10 Net 30
 - Annually
 - Collect on Delivery
 - Monthly
 - Net 15
 - Net 30
 - Quarterly

Figure 5.7: Optimized Layout of Customer FUI

The optimized UI shown in Figure 5.7, displays the functions for the image picture-box (add, remove, etc.) and address text-area (bold, italic, etc.) as buttons on the screen. In contrast, the version in Figure 5.4a provided these functions through a context-menu. Also, the payment terms list-box was substituted with a radio-button-group, which displays more items on the screen.

5.5 End-User Feedback for Refining the Adaptations

Keeping the end-users involved in the UI adaptation process, provides awareness of the adaptation decisions made by the system and the ability to override them when necessary. Therefore, RBUIS implements the feedback components of the Cedar Architecture (Figure 4.1). The final UI is transmitted to the client alongside a list of the applied simplification operations. We denote such operations by the UML interface called *Simplification* shown in both meta-models (Figure 5.2 and Figure 5.5). Our approach has two types of operations: feature-set minimization and layout optimization, identified by the *Role Ref* and *Task ID / Workflow ID* attributes respectively. The meta-model in Figure 5.2 shows *Reason Message* and *Is Reversible by User* as attributes of the *Simplification* UML interface (same for Figure 5.5). These attributes indicate the reason behind the simplification and whether it is reversible by the end-users.

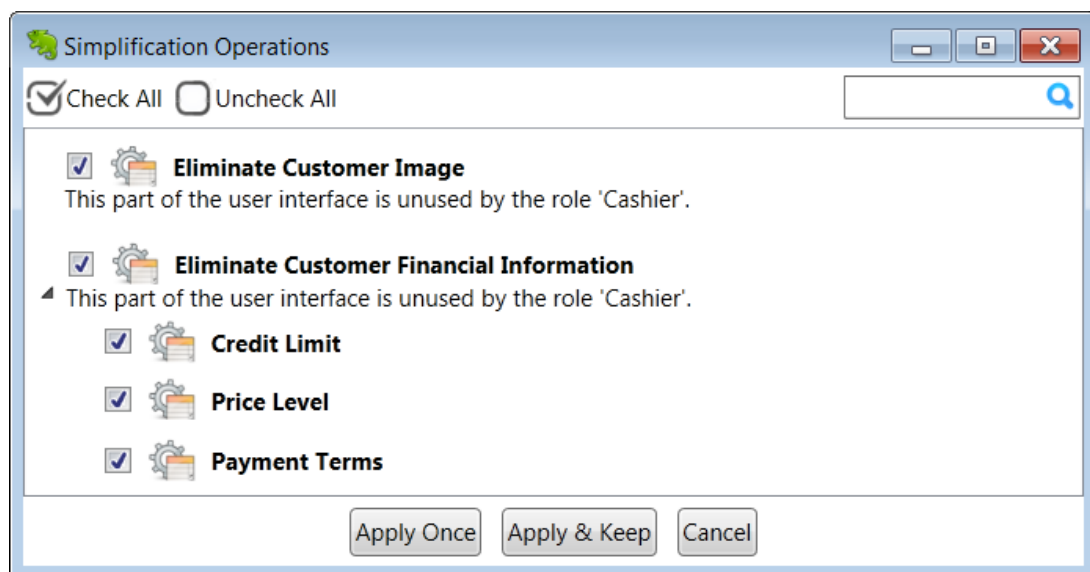


Figure 5.8: User Feedback Interface Showing Simplification Operations

Adapted UIs show a chameleon icon in their top right corner as illustrated in Figure 5.4b, and Figure 5.7. The end-users can click this icon to display a list of the applied adaptation operations similar to the one illustrated in Figure 5.8. Afterwards, the end-users can uncheck any reversible operation (feature-set minimization or layout optimization) and apply the changes for one time only or for future use as well. Furthermore, with layout optimizations, the end-users can choose from possible alternatives. This is achieved by assigning workflows to *Workflow Groups* as shown by the meta-model in Figure 5.5. Workflows in the same group could serve as alternatives. For example, a group can encompass several workflows each of which adapts the selection widget to one of the following types: combo-box, list-box, radio-buttons, etc.

After the end-user applies the changes, a request is sent to the server to re-simplify the UI and exclude the operations that were unchecked. In case the end-user decides to keep the changes for future use, the feedback would get stored, and he or she will gain access to an alternative version of the UI. The example operations illustrated in Figure 5.8 are related to the simplified UI in Figure 5.4b. The operations inform the end-user that the UI parts pertaining to the financial information and image were eliminated because they are unused by his or her role (Cashier). In this example, if the end-user unchecks both operations and applies the changes, the simplified UI in Figure 5.4b will revert back to the original version in Figure 5.4a. If an operation is set to be *irreversible by users*, for example due to security reasons, the check-box would become disabled and a message provides a notification of the reason. If a feature depends on other disabled features, the end-user is informed that these features should be enabled as well. The dependency is determined from the CTT temporal operators, and is defined in the meta-model shown in Figure 5.2 through the recursive relationship *Depends On* in the *Task* class.

5.6 Managing Trade-Offs in Multi-Aspect Adaptive UIs

Supporting trade-off analysis among several potentially conflicting aspects affecting the same UI factors helps in producing optimal UIs in an ever-changing context-of-use. The trade-off analysis technique presented in this section is a realization of the Cedar Architecture's *Trade-off Manager* (Figure 4.1).

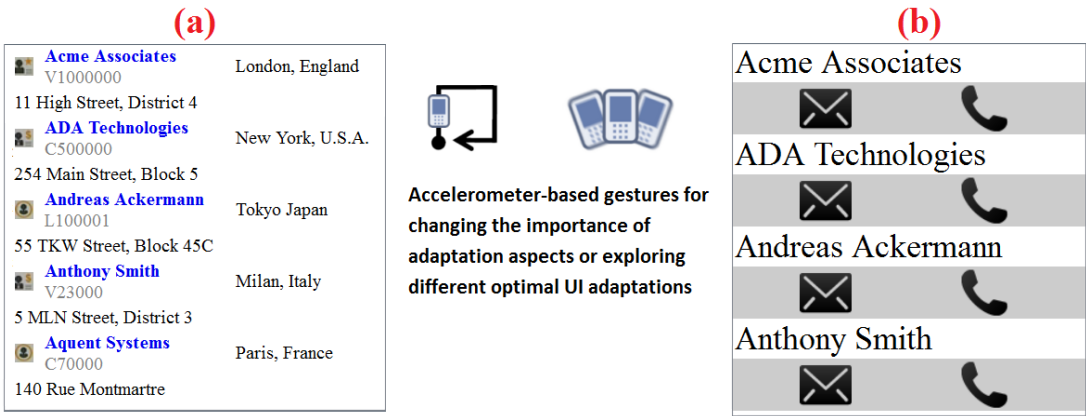


Figure 5.9: Business Partners Mobile User Interface based on the SAP Business One ERP Mobile Application (a) (Detail > Mobility) = Low Accessibility of Functions, Medium Font-Size, High Information Density (b) (Mobility > Detail) = High Accessibility of Functions, Large Font-Size, Low Information Density

Enterprise applications contain many UIs, which can be adapted based on multiple aspects. The example illustrated in Figure 5.9, presents two versions of a business partner’s mobile UI. When the end-user is in a state of low mobility such as: sitting or standing, the UI version shown in Figure 5.9a, provides more details while maintaining readability. Yet, in a state of higher mobility such as: walking or running, the UI version shown in Figure 5.9b, provides better readability and easier control.

By supporting aspect-level trade-off, innovative feedback techniques such as accelerometer-based gestures could be used on a mobile phone or a tablet for augmenting the importance of one aspect over another hence prompting the UI to adapt to the changing context-of-use. On a desktop computer, the end-users can use sliders for changing the level of importance of each adaptation aspect.

5.6.1 Trade-Off Analysis Technique

Multi-aspect trade-off analysis is required when multiple adaptation aspects simultaneously impact the same UI factors. We rely on goal models, as shown by the meta-model in Figure 4.3 (Chapter 4), to represent aspects and factors as goals and operations. Crisp goals are simply sorted by priority and the top goal is fulfilled by applying the adaptations relevant to the operations associated with it. This section focuses on fuzzy goals, for which a combination of Pareto

analysis and cost functions is used to determine the adaptations to be applied. Using preferences that compare situations under Pareto optimal conditions is considered promising for performing trade-off analysis in self-adaptive systems (Cheng et al. 2009). Pareto analysis and cost functions can compute trade-offs in multi-dimensional optimization problems and have been used in domains such as: economics and computer communications. The following example shows how we apply these concepts to adaptive UIs.

Let us consider a basic enterprise application scenario where trade-off analysis is required among three adaptation aspects: visual impairment, computer literacy, and device. In this example, we shall use three sub-aspects: medium vision user (MVU), novice user (NVU), and tablet user (TBU). We consider these adaptation aspects to have an impact on four UI factors: accessibility of functions (AF), widget grouping (WG), selection widget (SW), and font-size (FS). We should note that our approach is capable of accommodating more adaptation aspects and factors.

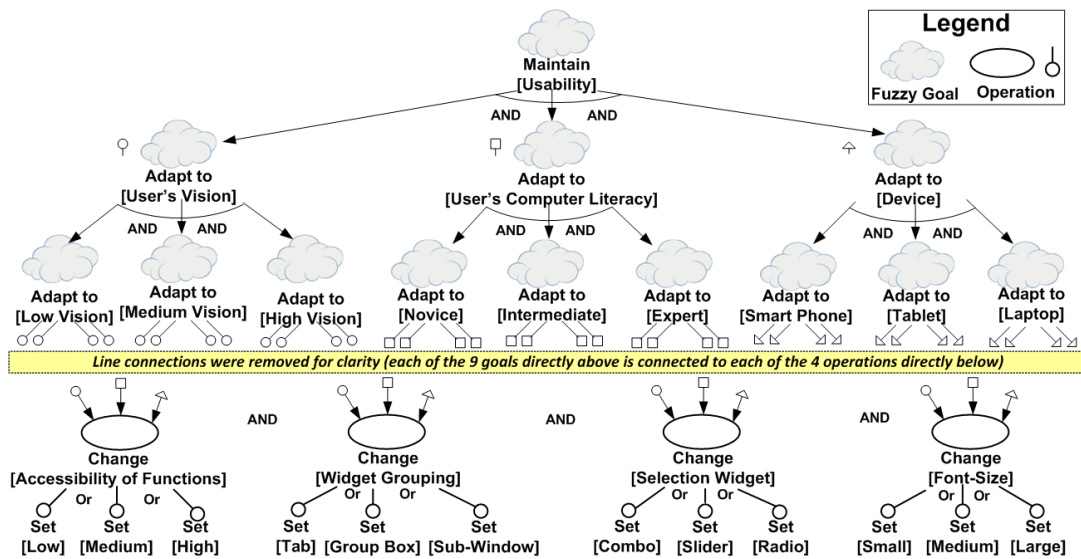


Figure 5.10: Multi-Aspect/Factor Adaptive UI Goal Model Example

This scenario can be represented by the goal model shown in Figure 5.10. We should note that the goal models, which we adopted for representing UI adaptation aspects and factors, are not meant to be used by UI designers. These models will be used by technical stakeholders, for example IT personnel, in the post deployment phase of enterprise applications.

The goal model in Figure 5.10 has costs associated with each fuzzy-goal/operation as represented by the cost matrix in Table 5.1. This matrix is populated with a cost for each adaptation aspect/factor combination. For example, end-users with medium vision prefer larger font-sizes hence the costs under the font-size factor were given the following values: Large = 1, Medium = 2, and Small = 3. Gajos & Weld (2005) suggested preference elicitation for populating such costs. We think that by combining preference elicitation with expert knowledge and data from studies, the time for populating the matrix could be reduced and the quality of its values could improve. Yet, our focus here is not on the means of populating the costs but rather on the technique for performing the trade-off analysis.

Table 5.1: Cost Matrix Example for Multi-Aspect Trade-Off Analysis

		Medium Vision User (MVU)	Novice User (NVU)	Tablet User (TBU)
Accessibility of Functions (AF)	Low	3	3	1
	Medium	2	1	2
	High	1	2	3
Widget Grouping (WG)	Tab-page	2	2	1
	Group-box	1	1	2
	Sub-window	3	3	3
Selection Widget (SW)	Combo	2	3	1
	List	3	2	2
	Radio	1	1	3
Font-Size (FS)	Small	3	1	1
	Medium	2	1	2
	Large	1	1	3

The number of possible sets of factors resulting from the cost matrix is given through a Cartesian product as follows:

$$|A \times B| = \{ (a, b) : a \in A, b \in B \}$$

Equation 5.1: Cartesian Product in a General Form

The Cartesian product of the factors shown in Table 5.1 is calculated as follows:

$$\begin{aligned}
 CPF &= |AF \times WG \times SW \times FS| \\
 &= |AF| \cdot |WG| \cdot |SW| \cdot |FS| \\
 &= 3 \times 3 \times 3 \times 3 = 81 \text{ Sets of Factors}
 \end{aligned}$$

Equation 5.2: Cartesian Product of Example Adaptation Factors

The cost: $Cost(A, F)$, is calculated per adaptation aspect A and set-of-factors F , $\forall F \in CPF$ as follows, where n is the number of factors in F :

$$Cost(A, F) = \sum_{k=1}^n F_k$$

Equation 5.3: Cost as a Function of an Adaptation Aspect and a Set of Factors

For example, the set of adaptation factors: Accessibility of Functions = *Medium*, Widget Grouping = *Group-box*, Selection Widget = *Radio*, and Font-Size = *Small*, has the following costs per adaptation aspect: Cost (Medium-Vision User) = 7, Cost (Novice User) = 4, Cost (Tablet User) = 8.

To find the optimal sets of factors from which one set can be selected, the Pareto Front is calculated based on: $Cost(A, F)$. We rely on an existing MATLAB algorithm (Cao 2008), which works on the entire design space and performs the calculation in $r \times n \times m$ for an $n \times m$ problem, where r is the size of the final Pareto Front. Other existing approaches that use machine-learning to calculate the Pareto Front by sampling a fraction of the design space can also be employed with significantly large problems. However, considering the number of aspects and factors that could be encountered in enterprise application UIs, the algorithm that we employed is efficient enough. Using the cost matrix from Table 5.1 as an example, this algorithm determined the Pareto Front in a few milliseconds.

In our running example, the Pareto Front includes 11 Pareto optimal elements (sets of factors) out of the initial 81 produced by the Cartesian product. The result is illustrated in Figure 5.11, which depicts the elements on the Pareto Front as squares and the ones outside it as circles.

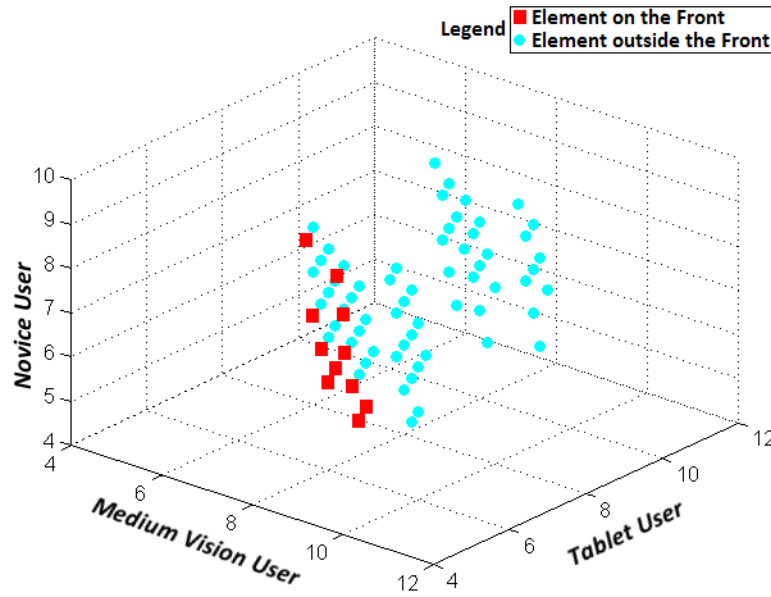


Figure 5.11: Pareto Front for Multi-Aspect Trade-Off

The Pareto Front represents interesting alternatives. However, when it is time to select one element (set of factors); additional information is given as a cost function that could be applied to all the Pareto Optimal elements to determine the one with the lowest cost. The cost function is given as follows, where N is the number of aspects, F_p is an element on the Pareto Front, and WA_k is the weight given by the end-user to an aspect A_k :

$$Cost(UI, F_p) = \sum_{k=1}^N \frac{Cost(A_k, F_p)_k}{WA_k}$$

Equation 5.4: Cost as a Function of a UI and an Element on the Pareto Front

A lower weight indicates that the aspect is more important. Hence, the weights are used with a division operation to decrease the cost of a set of factors F_p for the aspects with the lower importance in favor of those with the higher one. Initial weights supplied to the cost function, can be specified on the operations in the goal model. Each goal model will have a cost function for calculating a UI's cost with respect to a set of operations. End-users can augment the preference of one aspect over another using a feedback mechanism, for

example mobile gestures as shown in Figure 5.9. Alternatively, since the Pareto analysis minimizes the number of interesting UIs, end-users can explore these UIs visually and select the one that is the most satisfactory. Both approaches allow the end-users to evaluate the fully-adapted UI instead of just one factor against another. Such evaluation is considered useful for assessing adaptations that affect the global UI design because it presents the end-users with a UI showing the final and complete set of adaptations (Gajos & Weld 2005).

Determining the Pareto front before finding one optimal solution provides end-users with the ability to explore a small set of Pareto optimal UIs. The cost matrix can be populated with costs based on data from different sources such as: user-models obtained from on studies, expert knowledge, etc. Exploring the Pareto optimal UIs serves as a means of feedback for end-users to override these costs. If the costs were to be applied as is, by directly providing the end-users with the optimal UI, the Pareto front calculation can be skipped and the second cost function in Equation 5.4 would be directly applied to the sets of factors in the Cartesian product.

An excerpt from our algorithm for analyzing multi-aspect trade-offs before adapting the UI, is presented as pseudo-code in Listing 5.5. The running time of the algorithm is established to be polynomial: $O((m \times n) + j + (k \times m \times p) + (r \times m \times s) + (l \times s) + l \times \log l)$, where **m** = number of aspects, **n** = number of operations related to aspects, **j** = number of distinct factors, **k** = sets of operations in Cartesian product, **p** = number of operations in a Cartesian product set of operations, **l** = number of elements on the final Pareto Front, and **s** = number of costs associated with an element on the Pareto Front.

The complexity of the user interface does not affect the scalability of this algorithm (Listing 5.5) considering that it only operates on the goal model. The part that is affected by the UI's complexity is the execution of the workflows on the CUI models. The performance and scalability of the workflows were measured with complex real-life enterprise applications user interfaces and are presented as part of the evaluation in Chapter 7.

Listing 5.5: Analyze Multi-Aspect Trade-Offs and Adapt UI (Excerpt)

```

1. AdaptUI(GoalModel, Aspects[], CUIModel) {
2.     //Populate the Cost Matrix from the Goal Model
3.     var costMatrix
4.     foreach a in Aspects {
5.         operations[] ← GetOperations(GoalModel, a)
6.         foreach(o in operations) { costMatrix.Add(a, o, o.getCost(a)) }
7.     }
8.     //Group the Operations by Adaptation Factor
9.     var operations ← costMatrix.getOperations()
10.    distinctFactors[] ← operations.Select(o => o.Factor).Distinct()
11.    List<[]> operationsByFactor
12.    foreach(f in distinctFactors) {
13.        operationsByFactor.Add(operations.Select().Where(
14.            o => o.Factor = f))    }
15.    //Compute the Cartesian Product
16.    List<[]>cartesianProduct ← GetCartesianProduct(operationsByFactor)
17.    //Compute Aspect Costs for each Set of Factors in Cartesian Product
18.    var aspectCosts
19.    foreach(setOfOperations in cartesianProduct) {
20.        foreach a in Aspects {
21.            aspectCost ← 0
22.            foreach(o in setOfOperations) { aspectCost += o.GetCost(a) }
23.            aspectCosts.Add(a,setOfFactors,aspectCost)
24.        }
25.    }
26.    var optimalUI, setsOfFactors[] ← cartesianProduct
27.    if (FeedbackIsRequired) { //Compute the Pareto Front
28.        setsOfFactors ← GetParetoFront(aspectCosts) }
29.    //Compute Costs of Pareto Optimal Elements using Aspect Weights
30.    foreach(e in setsOfFactors) {
31.        foreach(aspectCost in e.AspectCosts)
32.            {e.Cost += aspectCost.Cost/Aspects.getWeight(aspectCost.Aspect)}
33.    }
34.    if (FeedbackIsRequired) { //Ask the User to Select an Optimal UI
35.        optimalUI ← PresentUserWithUIVariations(setsOfFactors) }

```



```

36.     else { //Get the Element with the Lowest Cost
37.         optimalAdaptation ← setsOfFactors.OrderBy(e => e.Cost).First();
38.         //Execute the Adaption Workflows on the CUI Model to Adapt the UI
39.         adaptationWorkflows[] ← LoadWorkflows(optimalAdaptation.Factors)
40.         optimalUI ← CUIModel
41.         foreach(w in adaptationWorkflows) { w.Execute(optimalUI) }
42.     }

```

5.6.2 Multi-Aspect Trade-Off Analysis Example

The Pareto-optimal sets of factors, which we obtained from calculating the Pareto Front depicted in Figure 5.11, are listed in Table 5.2. The costs in the columns representing the aspects: Medium Vision User (MVU), Novice User (NVU), and Tablet User (TBU), were calculated using Equation 5.3.

Table 5.2: Pareto Optimal Sets of Factors and their Costs for the Adaptation Aspects

#	Selection Widget (SW)	Widget Grouping (WG)	Font-Size (FS)	Access. of Functions (AF)	Medium Vision User (MVU)	Novice User (NVU)	Tablet User (TBU)
1	Radio	Group-box	Small	Medium	7	4	8
2	Radio	Group-box	Medium	Medium	6	4	9
3	Radio	Group-box	Large	Medium	5	4	10
4	Radio	Tab-page	Small	Medium	8	5	7
5	Combo	Group-box	Small	Medium	8	6	6
6	Combo	Group-box	Medium	Medium	7	6	7
7	Combo	Group-box	Large	Medium	6	6	8
8	Radio	Group-box	Large	High	4	5	11
9	Combo	Tab-page	Small	Medium	9	7	5
10	Combo	Group-box	Large	High	5	7	9
11	Combo	Tab-page	Small	Low	10	9	4

Consider the three scenarios shown in Table 5.3 as an example. The cost function from Equation 5.4 is applied to select the most optimal set of factors for each one using the weights in Table 5.3 and the sets of factors from Table 5.2.

Table 5.3: Three Example Scenarios with Different Weights for the Adaptation Aspects

Scenario	Weight of Adaptation (WA_k from Equation 5.4)		
	Medium Vision User (MVU)	Novice User (NVU)	Tablet User (TBU)
S1	1	2	3
S2	3	3	1
S3	1.5	2.5	2

The costs of the sets of factors for each adaptation scenario are shown in Table 5.4. We can see that set of factors 8 (radio, group-box, large, high) is the most optimal for scenarios S1, as for scenario S2 both sets 9 (combo, tab-page, small, medium) and 11 (combo, tab-page, small, low) provide optimal results, and for scenario S3 set 3 (radio, group-box, large, medium) is optimal.

Table 5.4: Pareto Optimal Sets of Factors and their Costs for Three Scenarios (Showing the Lowest Costs in Dark Grey)

#	Selection Widget (SW)	Widget Grouping (WG)	Font-Size (FS)	Access. of Functions (AF)	S1	S2	S3
1	Radio	Group-box	Small	Medium	11.66	11.66	10.26
2	Radio	Group-box	Medium	Medium	11	12.33	10.1
3	Radio	Group-box	Large	Medium	10.33	13	9.93
4	Radio	Tab-page	Small	Medium	12.83	11.33	10.83
5	Combo	Group-box	Small	Medium	13	10.66	10.73
6	Combo	Group-box	Medium	Medium	12.33	11.33	10.56
7	Combo	Group-box	Large	Medium	11.66	12	10.4
8	Radio	Group-box	Large	High	10.16	14	10.16
9	Combo	Tab-page	Small	Medium	14.16	10.33	11.3
10	Combo	Group-box	Large	High	11.5	13	10.63
11	Combo	Tab-page	Small	Low	15.83	10.33	12.26

Consider as an example a UI common in ERP systems, for maintaining bank account information. The algorithm adapted this UI according to the optimal sets of factors indicated in Table 5.4 for scenarios S1, S2, and S3. The adapted

UI versions are shown in Figure 5.12. One can see that changing the weights can cause a small change (e.g., S1 versus S3) or a large one (e.g., S1 versus S2).

(S1)

(S3)

(S2)

Figure 5.12: Excerpt from a Bank Account Maintenance UI Example with Layout Adaptation Based on Multiple Aspects (Vision, Device, and Computer Literacy) Affecting Several Factors (Accessibility of Functions, Widget Grouping, Selection Widget, and Font-Size) with Multi-Aspect Trade-Off: (S1) Medium Vision > Novice > Tablet, (S2) Tablet > (Medium Vision = Novice), (S3) Medium Vision > Tablet > Novice

5.7 Refitting the UI's Layout After Adaptation

Adaptive UI behavior such as widget hiding and substitution could leave gaps and deformations in the layout, which are not aesthetically desirable and could increase the navigation time. We required a mechanism to maintain plasticity, denoting the UI's ability to adapt to the context-of-use while preserving its usability (Coutaz 2010).

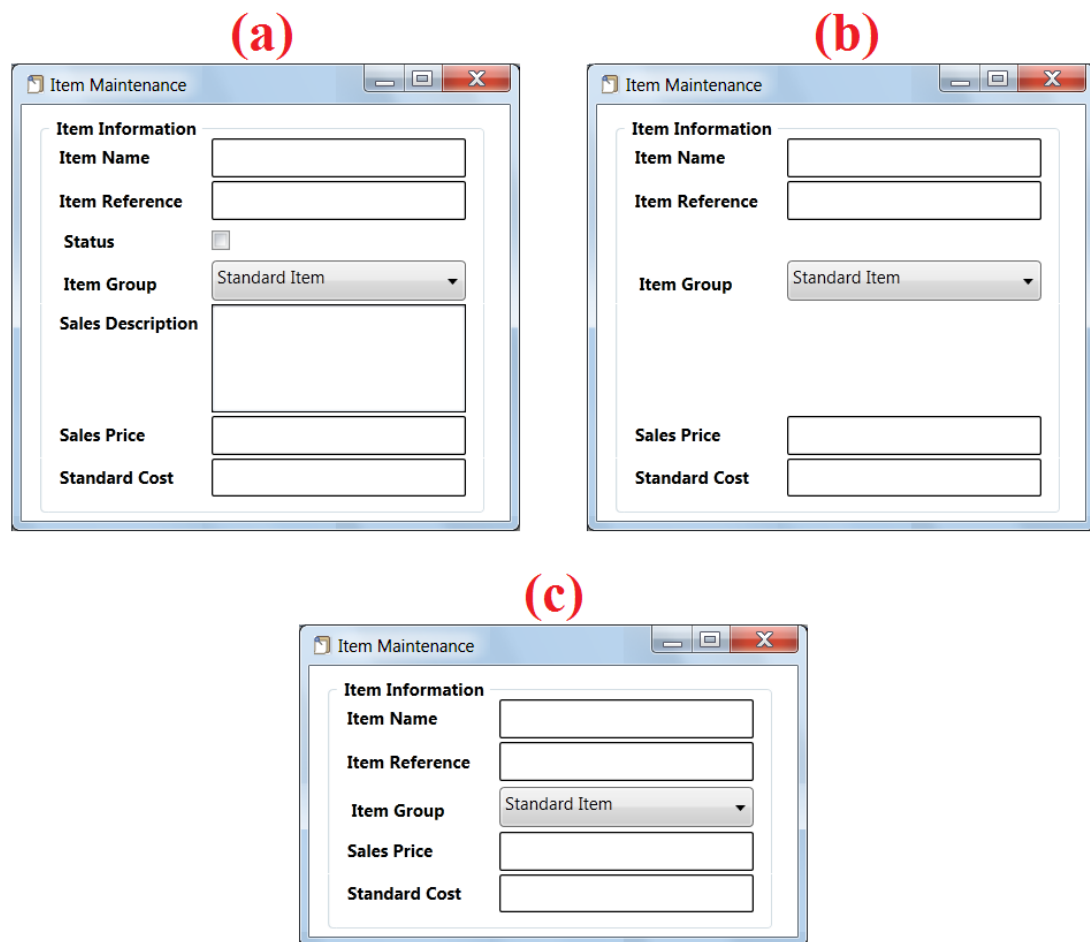


Figure 5.13: UI Refitting Example Using Relative Positioning
(a) Initial UI, (b) Minimized Feature-Set UI that Hides Widgets, and (c) Refitted Layout UI

We devised an algorithm for refitting the layout of an adapted UI based on the initial design choices made by UI designer. This algorithm fills the gaps and adjusts the widgets' positions based on their new sizes and initial locations chosen by the UI designer. For example, upon eliminating parts of the UI shown in Figure 5.13a to minimize its feature-set for a particular context-of-use as shown in Figure 5.13b, the layout refitting algorithm is responsible for removing the gaps. The example illustrated in Figure 5.13c shows how the widgets are pushed upwards beneath the closest widget.

Listing 5.6: Algorithm for Refitting the Layout of an Adapted UI (Excerpt)

```

1.  RefitTop(Controls [], StartingTop = 5) {
2.      //Divide the user interface widgets into lines
3.      List<[]> lines ← GetControlLines(Controls);
4.      if (lines.Count() = 0) { return; }
5.      //Set the top position of the widgets in the first line
6.      foreach (control in lines[0]) { control.Top ← StartingTop; }
7.      //Set the top position of the widgets in the remaining lines
      //based on the closest widgets in the line above
8.      for (counter = 1; counter < lines.Count(); counter++) {
9.          foreach (control in lines[counter]) {
10.             reverseLineCounter ← counter - 1, ctrsAbove [];
11.             while (ctrsAbove.Count() = 0 && reverseLineCounter >= 0) {
12.                 ctrsAbove ← select l from lines[reverseLineCounter]
13.                     where (l.Left > control.Left - l.Width &&
14.                         l.Left < control.Left + l.Width)
15.                     orderby l.Height descending;
16.                 reverseLineCounter--;
17.             }
18.             if (ctrsAbove.Count() > 0) {
19.                 ctrAbove ← ctrsAbove.First();
20.                 control.Top ← ctrAbove.Bottom + widgetMargin; }
21.             else { control.Top ← StartingTop; }
22.         }
23.     }
24. }

```

The part of our algorithm that pushes the widgets upwards is shown in Listing 5.6. The CUI controls are split into ordered lines and each widget is moved beneath the widget above it the previous line. We also created a similar algorithm for pushing the widgets to the left-hand side in order to fill the gaps.

5.8 Discussion

The Role-Based User Interface Simplification (RBUIS) mechanism provides a technical solution for minimizing the feature-set and optimizing the layout of

enterprise application UIs. It becomes part of a general purpose UI adaptation solution, which can be accessed by enterprise applications using an API that connects to a web-service. RBUIS answers research question **Q2**. It allows the definition of adaptive UI behavior both visually, using role assignments and workflows, and through code, using rules and scripts. Using the Cedar Architecture as a reference allows RBUIS to benefit from interpreted runtime models, which can support more advanced adaptations.

RBUIS fulfills the criteria we established in Chapter 2 (Table 2.2) for evaluating the state-of-the-art. It supports *direct and indirect adaptation*, adopts interpreted runtime models as a *modeling approach*, and supports the *levels of abstraction* suggested by CAMELEON. RBUIS supports *extensible visual and code-based adaptive behavior*. For example, layout optimization adaptive behavior can be represented by dynamic workflows, which can incorporate both visual and code-based programming constructs. RBUIS implements the *trade-off analysis* and *user-feedback* components proposed by the Cedar Architecture (Figure 4.1). It also supports the *extension of adaptation aspects and factors* using goal models. The *scalability* of the algorithms behind RBUIS was evaluated using a complexity analysis. In Chapter 7, we shall show how RBUIS can be *integrated into existing systems* and further demonstrate its *scalability* by applying it to real-life enterprise application UIs and load-testing its web-service.

The trade-off management technique that we presented for managing multi-aspect trade-offs answers research question **Q3**. As we mentioned in the literature review conducted in Chapter 2, there are existing techniques for adaptive UIs that do not consider trade-off management and others that perform it using cost functions for a limited and fixed number of aspects. A comparison between existing works and our trade-off management mechanism gives our approach the following advantages:

- Allow the cost function to accommodate an extensible number of adaptation aspects with weights that can be adjusted for specific tasks in evolving contexts-of-use: The basic example shown in Figure 5.9 demonstrates how the importance of certain aspects could change for the same end-user and UI when the context-of-use changes.

- Allow the end-users to explore a minimal set of interesting UIs: Applying a cost function to the sets of factors could yield several possibilities with close costs. We use Pareto analysis to give end-users the option of exploring a small set of Pareto optimal UIs.
- Allow easier high-level adaptation especially on hand-held devices: This ability is achieved by enabling the end-users to supply aspect weights through innovative means such as accelerometer-based gestures (Figure 5.9).
- Support both crisp and fuzzy adaptation goals: For example, UI security can be represented as a crisp goal fulfilled by a Boolean operation indicating whether or not a widget is visible, and usability could be represented by a set of fuzzy goals such as the ones depicted by the goal model in Figure 5.10.

Fully-automated UI generation, such as the approach used by Supple (Gajos et al. 2010), can degrade performance due to the need to explore all possible widget positions. On the other hand, manual approaches that require developing and maintaining multiple CUI versions are adopted in some commercial ERP systems such as SAP (Synactive GmbH 2010) but can be time consuming in terms of both development and maintenance. Our adaptation approach creates a balance between fully-automated UI generation and manual adaptation. It allows the definition of adaptive behavior, such as workflows, which can be applied to multiple UIs. Also, it automatically refits the adapted UI layout based on information such as the widgets' positions, from the initial design created by the UI designer.

We should note that trade-off management does not provide optimal choices for all UI factors. By nature trade-off management is a solution for a selecting an optimal combination of UI factors. Hence, some UI factors will be the most optimal while others are bound to be less optimal in case an end-user has contradicting preferences. For example, if an end-user prefers to have a large font-size and at the same time prefers the widgets to be grouped using group-boxes on the same page, the UI might have to scroll.

A limitation in the trade-off management technique is that we did not explore the various means for populating the cost matrix. However, the literature already presents different methods such as: user studies, expert knowledge, and preference elicitation, which could be used for obtaining such

costs. We think that a combination of these methods could potentially make the costs more accurate.

5.9 Chapter Summary

Role-Based UI Simplification (RBUIS) is a mechanism based on the Cedar Architecture for improving usability by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. RBUIS answers research question **Q2**. It presents meta-models and algorithms for supporting feature-set minimization and layout optimization for an extensible number of adaptation aspects and factors. RBUIS also supports the addition of adaptive behavior as needed visually in the form of role-assignments to task models and adaptive-behavior-workflows, and through code by using rules and scripts. End-users are given a feedback mechanism, which allows them to undo adaptations that were presented by the system or choose alternative ones when possible.

A trade-off management technique was also presented to answer research question **Q3**. This technique complements RBUIS by managing multi-aspect trade-offs using a combination of goal models, Pareto optimality, and cost functions. A detailed example was given to demonstrate the applicability of the technique and its advantages over existing solutions were discussed.

The algorithms of RBUIS and its trade-off analysis technique were shown to be polynomial through a complexity analysis. A layout refitting algorithm was also presented for eliminating gaps, which might appear in the user interface after it gets adapted.

In the next chapter, we present our integrated development environment (IDE) Cedar Studio, which supports the development of adaptive model-driven enterprise application UIs using RBUIS and following the Cedar Architecture. Cedar Studio offers stakeholders like software developers and IT personnel, with the necessary tools for creating and maintaining artifacts such as UI models and adaptive behavior.

6

Cedar Studio: An IDE Supporting the Development of Adaptive Model-Driven User Interfaces for Enterprise Applications

“If the only tool you have is a hammer, you tend to see every problem as a nail.”

—Abraham Maslow

Tools are necessary for supporting stakeholders interested in developing adaptive model-driven user interfaces. Enterprise applications in particular, require a tool that can be used by different stakeholders such as: software developers and IT personnel, during the development and (post)deployment phases. This chapter presents Cedar Studio, an integrated development environment (IDE) for devising adaptive model-driven UIs using RBUIS, which is based on the Cedar Architecture. Cedar Studio provides visual-design and code-editing tools for creating and maintaining artifacts such as UI models and adaptive behavior. It also provides the means for testing the UI adaptations.

6.1 Introduction

As we demonstrated in Chapters 4 and 5, the model-driven approach to UI development can serve as a basis for devising adaptive enterprise application UIs. This approach offers the possibility of applying different types of adaptations on the various levels of abstraction. However, practically implementing adaptive model-driven UIs requires tools that support the creation and maintenance of UI models and adaptive behavior. Existing tools lack many features required for supporting the development of adaptive model-driven enterprise application UIs. From a model-driven engineering perspective,

such tools should support the modeling, generation, and synchronization of all the UI levels of abstraction. Providing the ability to devise the adaptive behavior both visually and through code, allows such tools to support software developers as well as IT personnel. Furthermore, offering an IDE style UI could provide the necessary ease-of-use for managing the complex user interface and adaptive behavior artifacts of large-scale enterprise applications.

Cedar Studio is an integrated development environment (IDE), which supports the development of adaptive model-driven enterprise application user interfaces using RBUIS and is based on the Cedar Architecture. As we indicated in Chapter 4, UI adaptation techniques based on the Cedar Architecture are offered as a service, which can be consumed by Cedar Studio and technology specific APIs. Therefore, Cedar Studio can connect to the server-side layers, through a web-service, for managing the artifacts stored in the relational database on the UI Models and Adaptive Behavior layer (Figure 4.1).

The adaptations supported by Cedar Studio focus on UI simplification, which is offered by the RBUIS mechanism presented in Chapter 5. Tool support is offered for the parts of the Cedar Architecture and RBUIS, which are illustrated by Figure 6.1.

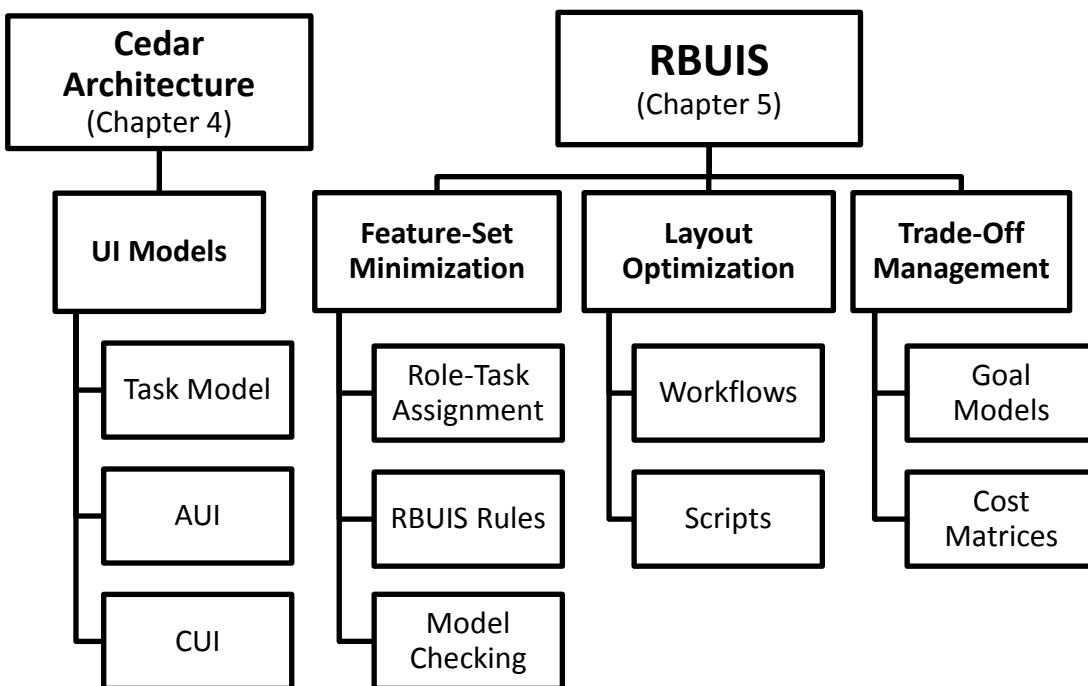


Figure 6.1: Parts of Cedar Architecture and RBUIS Supported by Cedar Studio

Cedar Studio provides software developers and IT personnel access to all the visual-design and code-editing tools in one place. It supports visual-design tools for the following artifacts: (1) task models (Figure 4.5), (2) domain models, (3) abstract UI (AUI) models (Figure 4.6), (4) concrete UI (CUI) models (Figure 4.7), and (5) goal models (Figure 5.10). It also supports automatic generation and synchronization between various UI levels of abstraction, and offers the possibility of making manual changes at any level. For example, the AUI can be generated from the task model and the CUI can be generated from the AUI.

A set of visual-design and code-editing tools, which are necessary for implementing adaptive UI behavior, are provided by Cedar Studio. These tools support: (1) visual adaptive-behavior-workflows (Figure 5.6) and (2) dynamic scripts for optimizing a UI's layout (Listing 5.4); (3) visual role-assignments (Section 5.3.2) and (4) code-based rules (Section 5.3.2) for minimizing a UI's feature-set to a particular context-of-use; and (5) SQL-based model-constraints for verifying manually created models (Section 5.3.4).

Cedar Studio can be used during the development and (post)deployment phases of a software application's lifecycle. The UI models are created at development time and the adaptive UI behavior is added at deployment time and can be modified at a later stage according to the needs of each enterprise.

Cedar Studio answers research question Q4, which we established in Chapter 3 as follows:

Q4: What integrated development environment can help software developers and IT personnel in developing and maintaining model-driven enterprise application UIs and adapting them with RBUIS?

6.2 Design Tools for User Interface Models

This section describes the design tools offered by Cedar Studio for supporting the creation and maintenance of the different UI models (task, AUI, and CUI).

6.2.1 Task Models

The design tool shown in Figure 6.2 supports visual composition of task models based on the ConcurTaskTrees (CTT) notation.

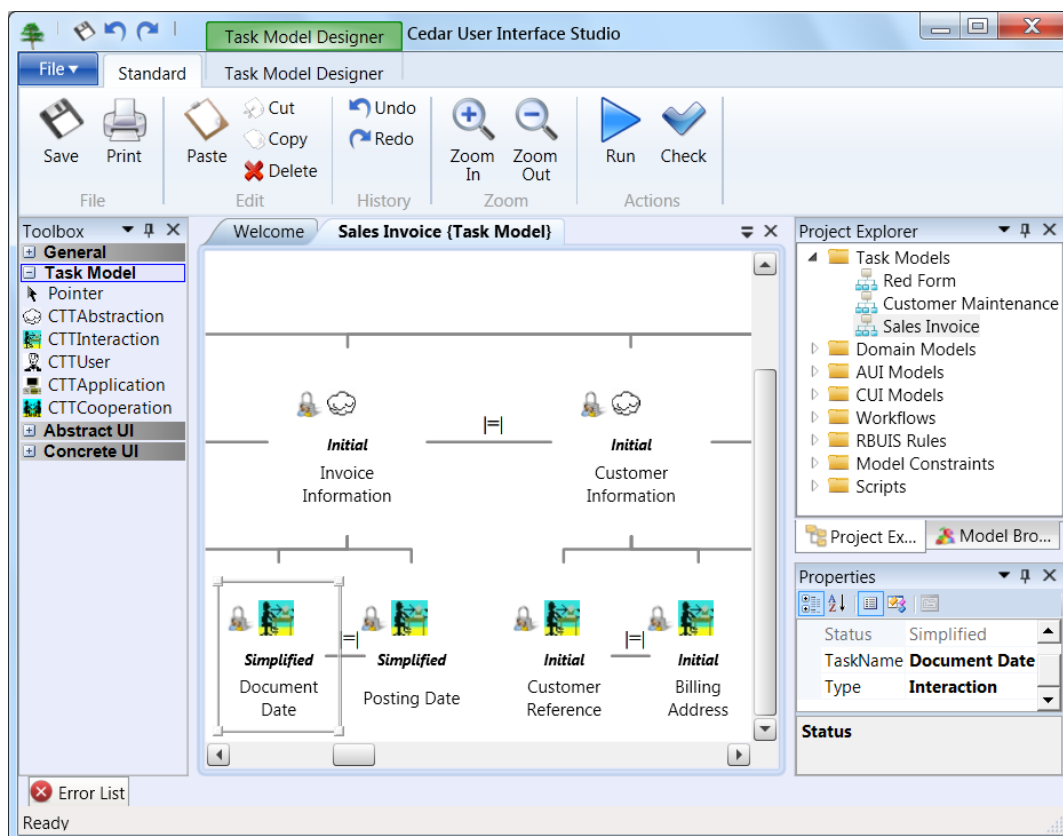


Figure 6.2: Task Model Design Tool

The importance of the task-model design tool is that it allows UI designers to visually design task models and allocate roles to them using the dialog shown in Figure 6.3, while maintaining the ability to allocate roles through more general code-based rules using the code editor shown in Figure 6.4. This visual and code-based combination for applying RBUIS in enterprise scenarios enhances the expressive match denoting the closeness between the means for applying design choices and the problem at hand (Olsen,Jr. 2007).

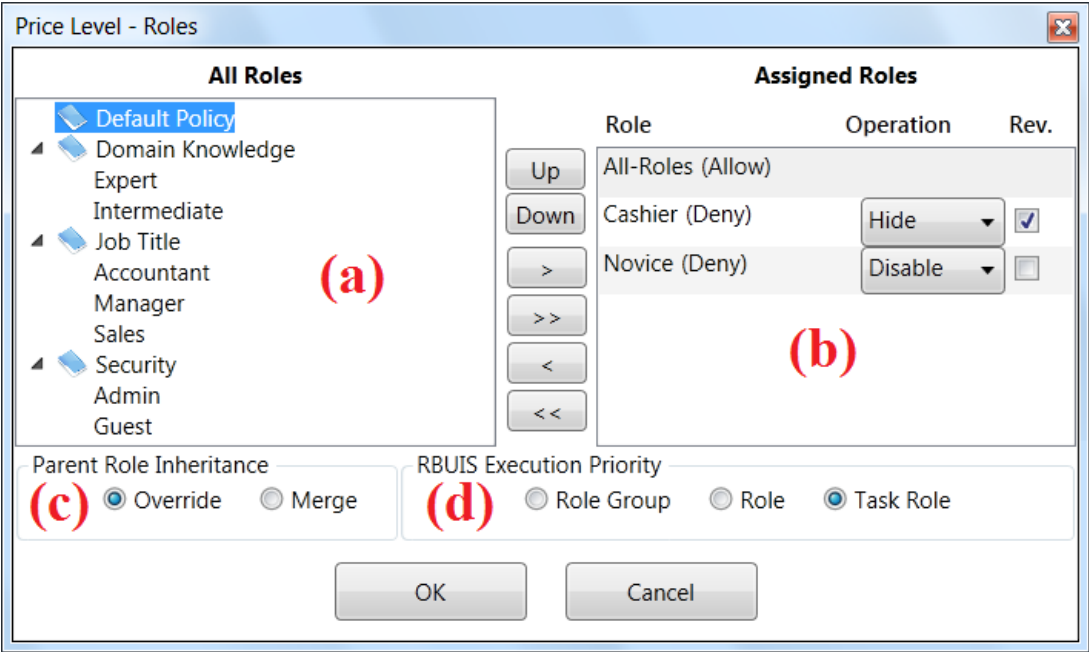


Figure 6.3: Visual Role Allocation

The task-model design-tool supports a tree layout algorithm, which can automatically adjust the presentation of large task models. Visual and code-based support is provided for the simplification process through role allocation to tasks. The lock-shaped button next to each task (Figure 6.2), allows a visual allocation of access rights using the UI shown in Figure 6.3. A default policy (“All-Roles”) is implicitly assigned to grant access to all the roles on any given task (Section 5.3.2). This policy can be overridden by explicitly assigning roles from different groups (Figure 6.3a) to each task. The concrete operation (e.g., hide, disable, etc.) and the ability to reverse it by the end-user are specified for each role (Figure 6.3b). A task can inherit or override roles assigned to its

parent task (Figure 6.3c). The order of each role can be changed to indicate its priority. The priority source can be explicitly indicated (Figure 6.3d).

The allocation of roles to tasks can also be done through SQL-based rules. These rules are written in the form of an SQL condition (RBUIS Rules in Section 5.3.2) conforming to the meta-models we presented in Chapters 4 and 5. This condition is assigned roles and allocated to the task models on which it should be executed. Cedar Studio provides a code-editor for these rules and supports the ability to validate the SQL syntax and display errors in an error list.

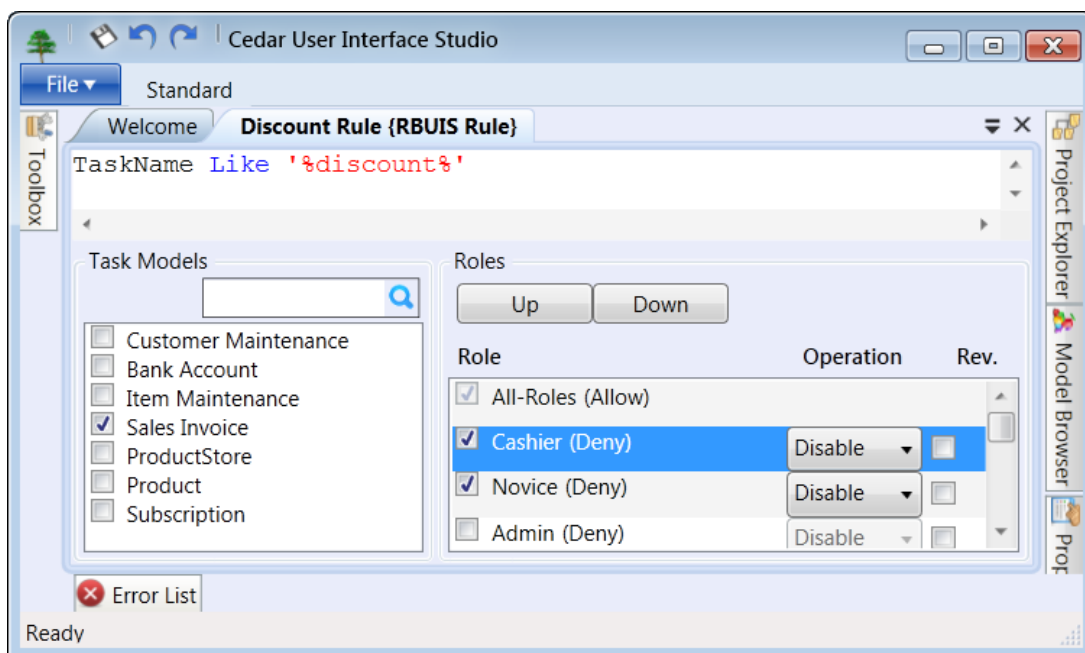


Figure 6.4: RBUIS Rules Code Editor

Model checking is required due to possible human errors in the allocation of roles to tasks (Section 5.3.4). Cedar Studio provides the code-editor shown in Figure 6.5 to support the composition of SQL-based model-checking constraints. A checklist allows the visual-assignment of these constraints to the task models on which they should execute by default. The erroneous tasks returned by the SQL statement are displayed in the error list to permit the technical stakeholders to identify them and fix the assignments in the appropriate task model.

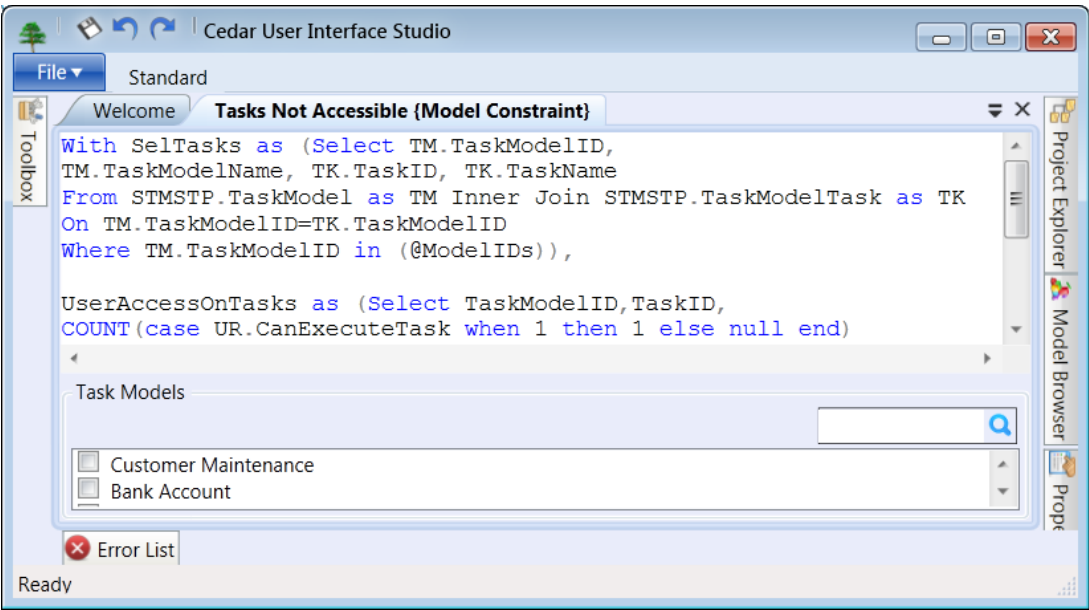


Figure 6.5: Model-Checking Constraints Code Editor

The second level of abstraction, namely AUI models, can be automatically generated from the task model. It is possible to visually override the default mapping using the UI shown in Figure 6.6, by allocating each task one or more AUI elements. This option exempts the designers from having to individually add, delete, or modify elements on the canvas after the generation is done.

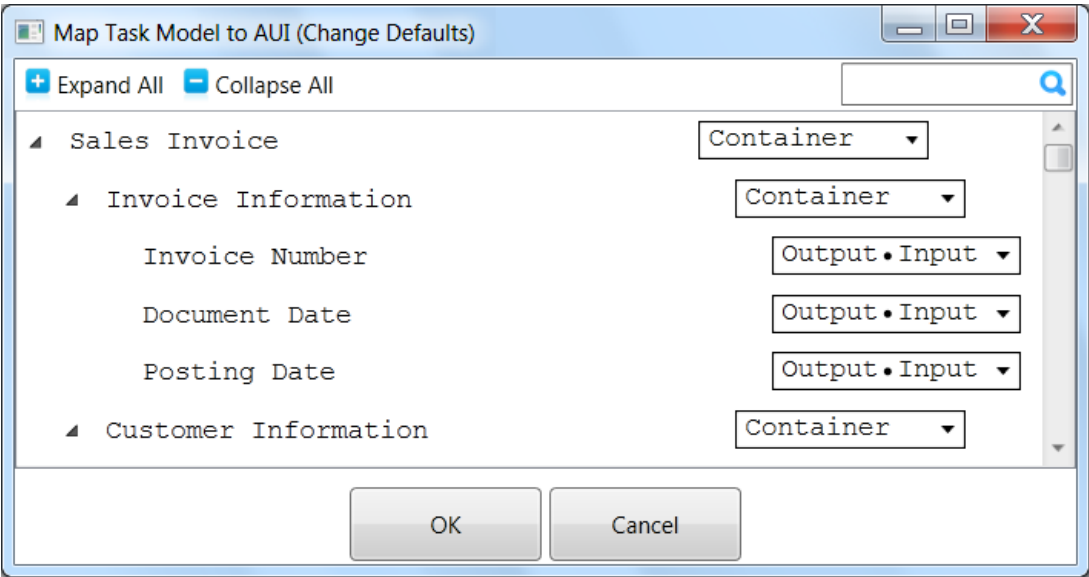


Figure 6.6: Mapping Task Model to AUI

6.2.2 Abstract User Interface Models

The generated AUI is easily modifiable through the visual-design tool shown in Figure 6.7. Simplicity is the main advantage of this tool, which supports the specification of AUIs with basic building blocks on a flow-style layout canvas that could be used by stakeholders with a limited technical knowledge.

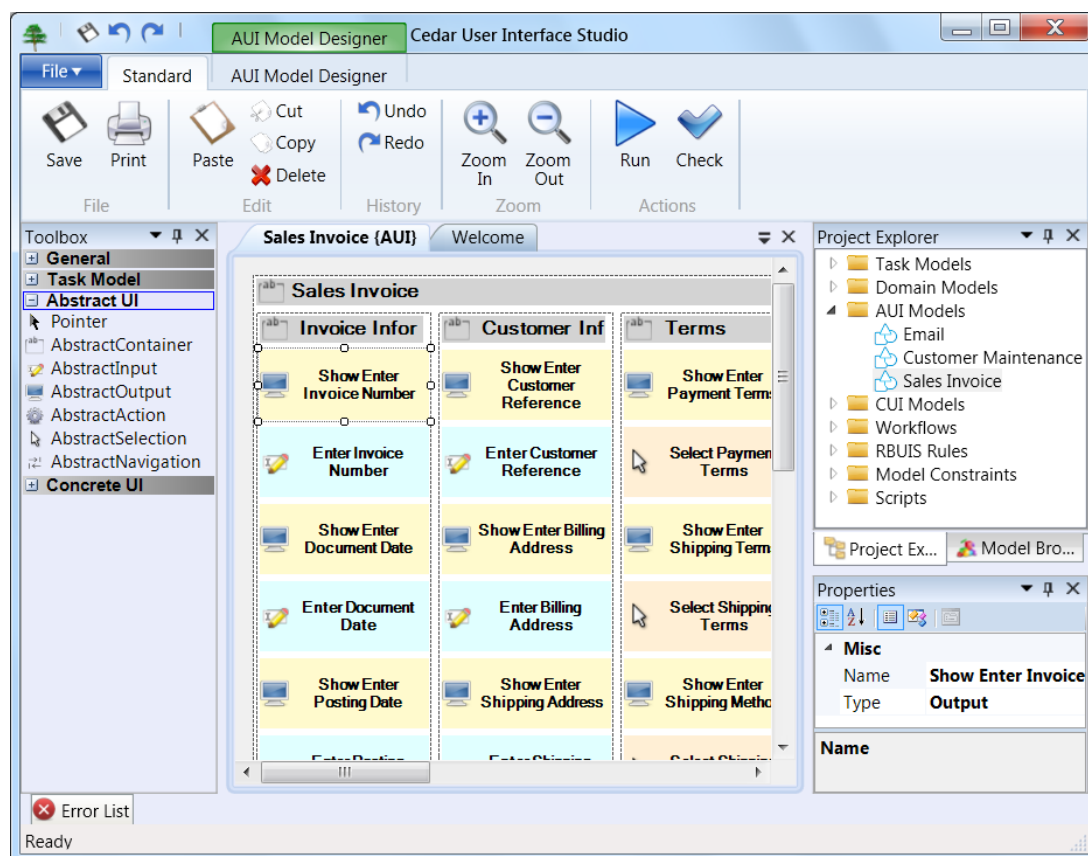


Figure 6.7: Abstract User Interface Design Tool

Since AUI models are a modality independent representation, the design canvas shows the elements as boxes with different names, icons, and colors. This tool allows AUI containers to be nested within one another and provides an easy-to-use flow-style design surface for visually manipulating the AUI elements. The properties box allows the modification of an element's name and type. As suggested by Pleuss et al. (2010), placeholder elements are used upon deletion to maintain the mapping between two models. The type of the placeholder can be switched to an AUI element type without affecting the

mapping. New elements can be added from the toolbar and then manually mapped to their related tasks in the task model.

CUI models can be automatically generated from AUI models, similarly to how AUI models are generated from task models. The interface shown in Figure 6.8 can be used for manually adjusting the default mappings.

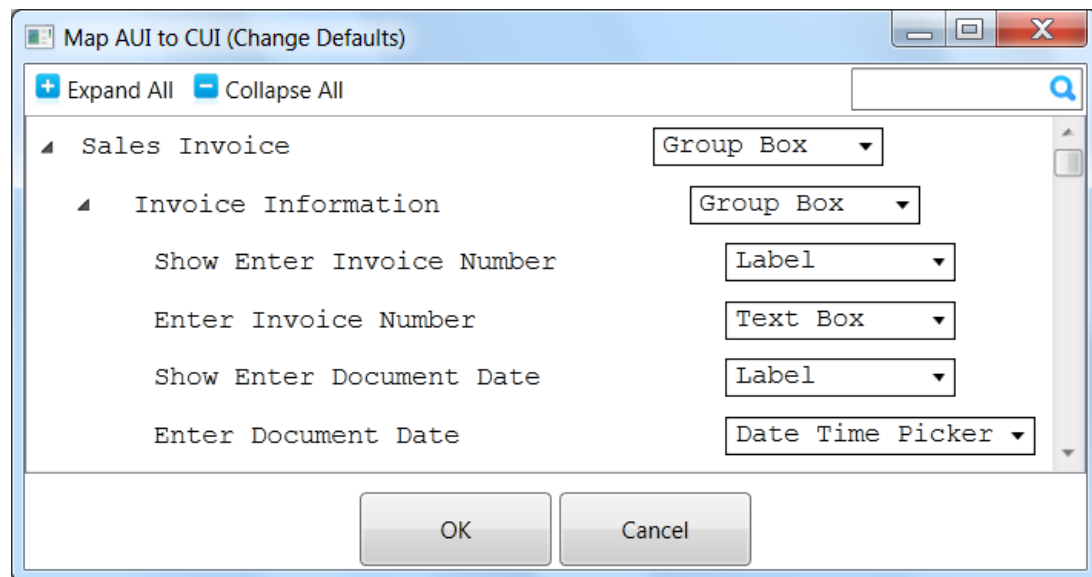


Figure 6.8: Mapping AUI to CUI

6.2.3 Concrete User Interface Models

The input of the human designer is highly desirable for achieving higher usability (Pleuss et al. 2010) through the manipulation of concrete objects rather than just an abstract representation (Demeure et al. 2009). Offering a robust CUI design tool to UI designers, helps them in providing their input on the look on feel of the UI. Visual user interface builders provide a graphical means for expressing graphical concepts, thereby maintaining a low threshold due to the reduction of the learning curve (Myers et al. 2000).

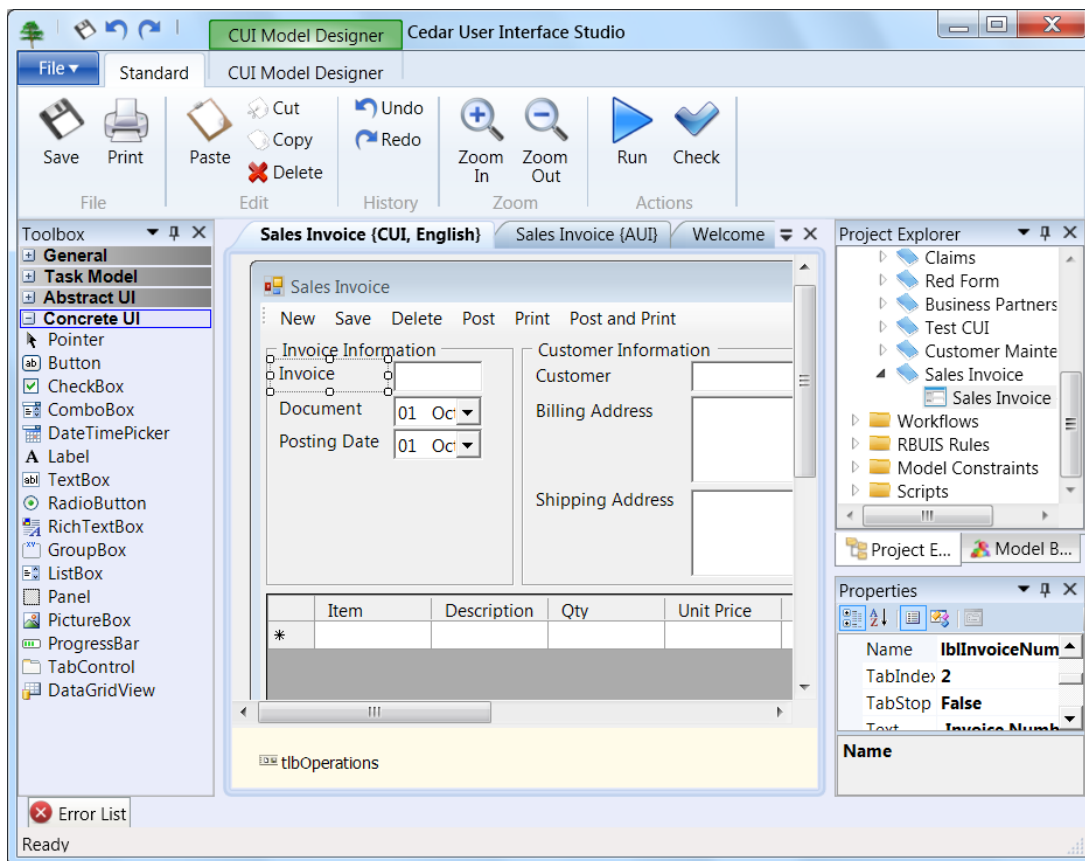


Figure 6.9: Concrete User Interface Design Tool

Cedar Studio provides the feature-rich CUI design tool shown in Figure 6.9 by integrating and extending the Windows-Forms design component of Visual Studio.NET. This component has been extensively tested through its usage in developing UIs for many enterprise applications. Similar to that of the AUI, the CUI design-tool supports placeholders upon deletion in addition to complete deletion of elements, which can be manually replaced and mapped to the AUI model. A rich toolbar of widgets is provided. It includes both basic widgets such as the date-time-picker and advanced ones such as the data-grid, which are required for devising enterprise application user interfaces.

6.3 Design Tool for Adaptive Behavior Workflows

Workflows are common in enterprise applications for representing business rules. Our approach utilizes workflows to represent adaptive behavior visually and through code (Sections 5.4.1 and 5.4.2). This approach provides both

developers and IT personnel the opportunity to implement this behavior using the visual canvas shown in Figure 6.10a. Similar to the task-model-design and role-assignment tools, the visual and code-based combination also enhances expressive match. Furthermore, providing expressive leverage by promoting reusability (Olsen,Jr. 2007) is achieved by supporting the integration of reusable visual-components and scripts.

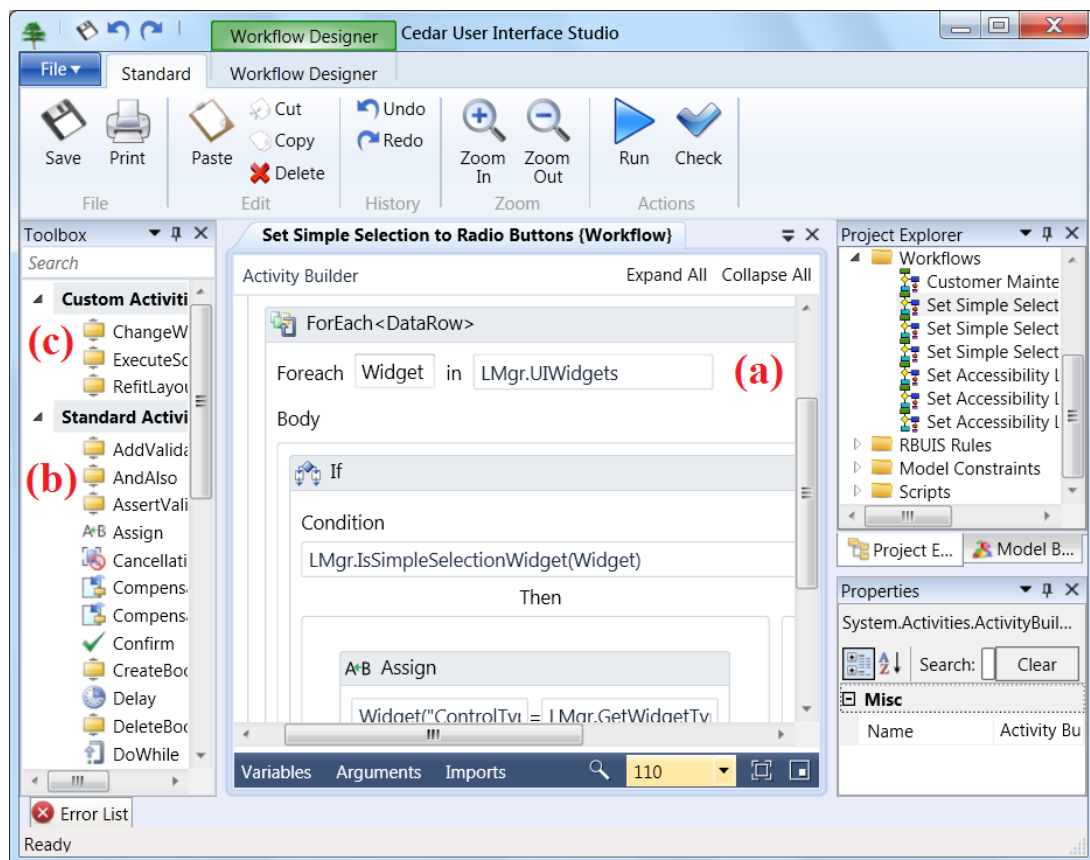


Figure 6.10: Adaptive Behavior Workflow Design Tool

Workflows are assigned roles and the CUI models to be executed on. We integrated the Windows-Workflow design tool of Visual Studio.NET in Cedar Studio. This tool provides a rich set of visual programming constructs (Figure 6.10b), which can be dynamically extended with custom activities (Figure 6.10c) written in C# or VB.NET. One of the extensions we have built supports the execution of adaptive behavior written in the Iron Python scripting language. Cedar Studio stores the workflows in an XML format, which allows them to be dynamically loaded and executed.

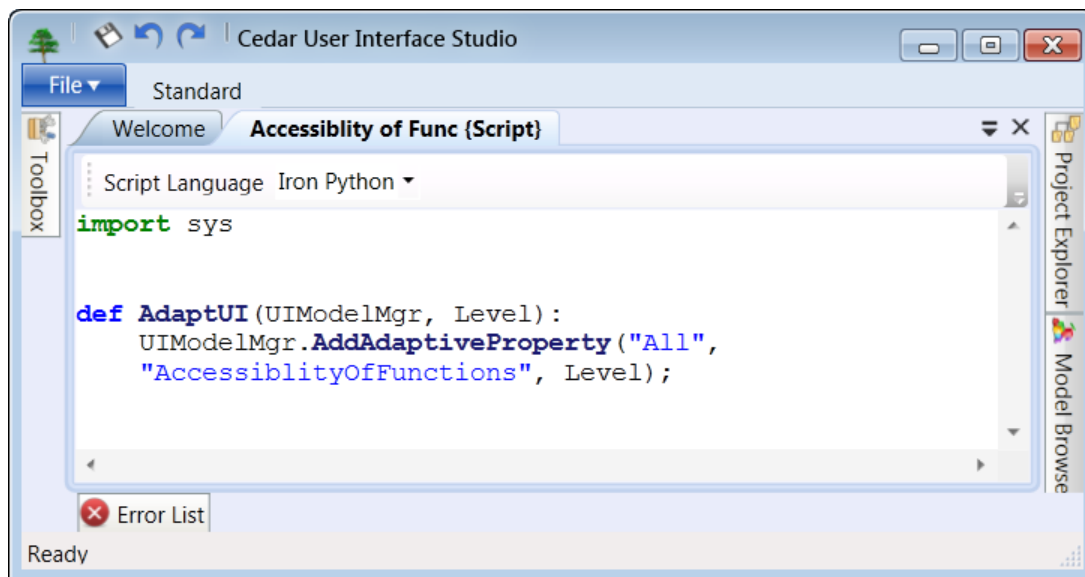


Figure 6.11: Dynamic Scripts Code Editor

The Iron Python script editor, illustrated in Figure 6.11, is supported by Cedar Studio. Scripts are created separately and can be called from within any workflow by selecting the script's name, specifying the method to call, and passing it the appropriate parameters. The entire process is done visually through the workflow design tool shown in Figure 6.10.

6.4 Design Tool for Goal Models

Cedar Studio supports a visual-design tool for goal models, which is shown in Figure 6.12. This tool allows stakeholders to define fuzzy and crisp goals, and operations in goal models, which are used as part of our trade-off management technique (Section 5.6, Chapter 5). As illustrated in Figure 6.12, each operation (displayed as ovals) on the lowest level of operations in the goal model, has a button above it. Upon clicking this button, Cedar Studio displays an interface for assigning the cost of that operation for each of the goals. These costs are used by our approach for populating the cost matrix, which is required for managing the trade-offs among the conflicting adaptation aspects.

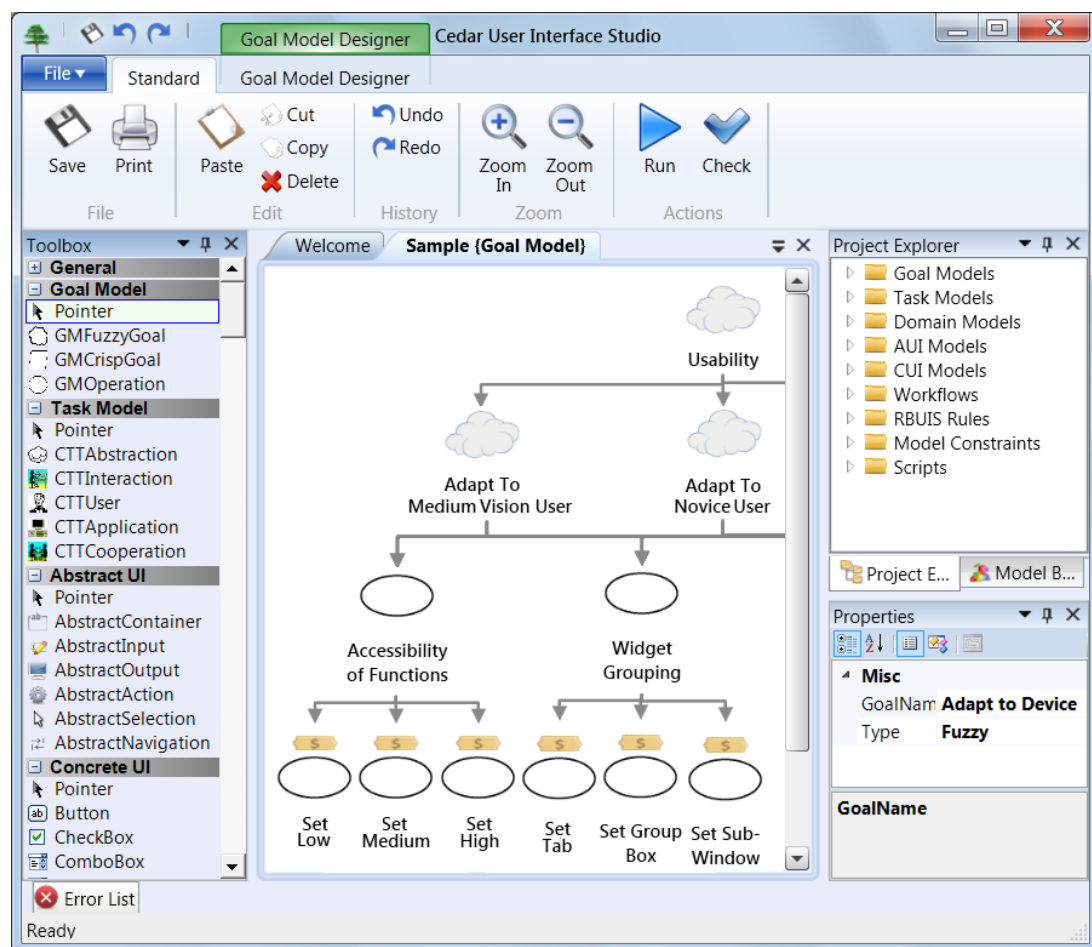


Figure 6.12: Goal Model Design Tool

6.5 Testing the Adapted User Interfaces

Developers can run the devised UIs with and without adaptations using the “Run” and “Run As” commands respectively. The “Run” command simply executes the initial version of the UI, whereas “Run As” issues a prompt for entering a user-identifier and executes the UI version corresponding to that user’s roles. This functionality allows stakeholders to test UIs and adaptive behavior from within Cedar Studio. Combining this feature with the previously described design tools, achieves flexibility in terms of supporting rapid design changes, which can be performed and evaluated by the developers (Olsen, Jr. 2007).

allocating the role “Cashier” to the appropriate tasks, applying the relevant adaptive behavior workflows, and running the UI with a user-identifier allocated the role “Cashier”, the version illustrated in Figure 6.13b will be displayed.

When the user’s role is modified, for example Cashier to Manger or Novice to Expert, the adaptation will dynamically change according to the new role. This conforms to the concept of multi-layer interface design (Shneiderman 2003).

6.6 Assessing Cedar Studio

Cedar Studio was assessed by constructing UIs based on examples from existing enterprise applications, and applying adaptive behavior to them. The UIs we constructed are data entry interfaces for managing: bank accounts (Figure 5.12), customers (Figure 5.4), inventory items, and sales invoices (Figure 6.13). The adaptations that we applied to these UIs represent practical scenarios, which could potentially occur in real-life enterprise systems.

Consider the initial sales invoice UI, shown in Figure 6.13a, as an example. The simplified version of this UI, shown in Figure 6.13b, is better suited for end-users allocated the role “Cashier”. The feature-set has been minimized to only show the fields required by cashiers, allowing end-users who are allocated this role to complete sales invoice transactions more efficiently. Additionally, the layout was optimized by switching combo boxes with radio buttons and showing a higher accessibility of functions, thereby allowing cashiers to control the UI with more ease through the point-of-sale touch screen. On the other hand, the simplified sales invoice UI for end-users allocated the role “Sales Officer” only removes the payments grid and totals fields, as illustrated in Figure 6.14, because sales officers require more functionality. Additionally, since sales officers use a mouse rather than a touch screen, the combo boxes are not replaced and a lower accessibility of functions is given.

A similar rationale was used when selecting adaptations for the other UIs that we constructed. By using Cedar Studio to construct these example UIs and adaptations, we were able to get a feel of this tool’s strengths, in addition to the points that could use further improvement.

Figure 6.14: Sales Invoice UI Simplified for the Sales Officer Role

One of the main observed strengths of using Cedar Studio in practice is in its AUI, CUI, and Workflow design-tools, which are based on existing mature Visual Studio components. The task model design-tool can be developed further to reach the same level of maturity and the code editors can be enhanced by providing additional functionality such as intelligent-sense. We should note that we do not claim Cedar Studio to be an industrial quality IDE. However, considering the advanced tools that it offers, it forms a strong starting point towards achieving this objective.

Cedar Studio fulfills the criteria we established in Chapter 2 for evaluating the state-of-the-art. This IDE offers *control over the UI* by allowing stakeholders to provide their input on all the *levels of abstraction* using visual-design tools. Cedar Studio also supports *visual-design and code-editing tools* for defining *extensible adaptive behavior and adaptation aspects and factors*. Its integrated testing, visual-design tools, and *IDE style UI* allow Cedar Studio to offer good *flexibility*. This IDE supports *automatic generation* between the UI models, and the mapping rules can be visually changed to offer better *predictability*.

In the previous sections, we described the advantages of Cedar Studio in terms of criteria such as: flexibility, expressive match, and expressive leverage. In this section, we assess Cedar Studio based on another set of criteria recommended for user interface development tools (Myers et al. 2000):

- **Threshold and Ceiling:** The “threshold” represents the difficulty in learning and using the tool, and the “ceiling” is related to how advanced the tool’s outcome can be. An ideal tool would have a low threshold and a high ceiling.
- **Path of Least Resistance:** Developers should be guided to construct the UI in an appropriate manner by making the right approach easier to follow than the wrong one.
- **Predictability:** Any automated approach provided by the tool should be predictable to the developers using it.
- **Moving Targets:** The tool should be able to keep up with the rapid developments in user interface technology.

Upon designing and developing Cedar Studio, we tried to meet the abovementioned criteria as much as possible.

It might not be feasible to achieve low threshold and high ceiling in all cases. This point is due to the learning curve created by any additional features, which allow a tool to produce more advanced outcomes. However, we aimed towards achieving a balance between *threshold and ceiling*. We integrated automated generation and synchronization between models (low threshold), alongside the possibility of conducting manual adjustments (high ceiling). Furthermore, if developers understand the semantics of the meta-model they can use the visual-design tools to produce an advanced outcome (medium threshold / high ceiling). In the cases where coding could be used, a visual-design tool alternative was provided such as: visual workflows instead of scripts, and visual role-assignments instead of code-based rules. Also, the language most familiar to developers and IT personnel was chosen in the case of using SQL instead of OCL for model verification.

The *path of least resistance* is maintained by allowing developers to easily apply the model-driven approach. The automated generation of UI models and the mapping between them, saves the time required to perform these operations manually. The automatic generation preserves *predictability* by allowing developers to customize the default mappings between the different model elements (e.g., abstract input to text-box). Furthermore, the support for visual-

adjustment and resynchronization provides an easy way to customize what was automatically generated.

Concerning the *moving targets* criteria, the model-driven approach supported by Cedar Studio was initially created to absorb the effect of changes in technology and requirements. The model-driven approach allows our IDE to be independent from presentation technologies, and to evolve more easily alongside them. If new techniques for building UIs or even new UI types emerge in the future, the use of models is a good approach to cope with such changes. The use of models makes it possible to rely on existing abstract representations to regenerate different types of concrete user interfaces.

6.7 Chapter Summary

This chapter presented an overview of Cedar Studio, an IDE for supporting different stakeholders such as: software developers and IT personnel, in developing and maintaining adaptive model-driven enterprise application user interfaces. The supported adaptive behavior is primarily targeted at the simplification of enterprise UIs by using RBUIS to minimize a UI's feature-set and optimize its layout based on the context-of-use.

Cedar Studio answers research question **Q4**, which we established in Chapter 3, by providing technical stakeholders with visual-design and code-editing tools for creating and managing UI models and adaptive behavior. It also supports integrated testing of the devised adaptive behavior by running the developed UI from within the IDE. We assessed Cedar Studio conceptually based on a set of criteria suggested in the existing literature, and practically by developing example adaptive enterprise application user interfaces.

Cedar Studio can be observed in operation through demonstration videos, which are available online¹⁰.

The contributions made in this thesis, are evaluated in the next chapter from both the technical and the human perspectives.

¹⁰ Cedar Studio Demonstration Videos: <http://adaptiveui.pierreakiki.com>

7

Evaluating the Contributions from the Technical and Human Perspectives

“One of the great mistakes is to judge policies and programs by their intentions rather than their results.”

— Milton Friedman

The Cedar Architecture and RBUIS are evaluated from a technical perspective by defining and applying software engineering metrics to measure several of their characteristics. To perform the technical evaluation, we integrated RBUIS in an existing open-source enterprise application called Apache Open For Business (OFBiz). The integration method is based on the Cedar Architecture. The contributions presented in thesis are also evaluated from a human perspective. We evaluated our approach based on the opinions of industry experts and data from real-life projects. We conducted usability studies to test whether UI simplification significantly improves end-user efficiency, effectiveness, and satisfaction in real-life enterprise application scenarios.

7.1 Introduction

We presented the contributions of this thesis namely the Cedar Architecture, RBUIS, and Cedar Studio in Chapters 4, 5, and 6 respectively. This chapter presents an evaluation of these contributions that covers both the technical and human perspectives. The work presented in this chapter answers the evaluation research questions, which were established in Chapter 3.

One of the gaps that we identified in Chapter 2 is related to the ability of adaptive UI solutions to integrate in existing systems. In this chapter, we

elaborate more on this point by discussing the strengths and shortcomings of existing approaches including: toolkits, aspect-oriented programming, design-time model-driven, and runtime model-driven. We integrated RBUIS into an existing open-source enterprise application called Apache Open For Business (OFBiz) using an integration method based on the Cedar Architecture. We established and applied several technical metrics, which allowed us to answer research question Q5 that was presented in Chapter 3 as follows:

Q5: Does the devised UI adaptation approach (Cedar Architecture and RBUIS) integrate in existing enterprise applications without causing major changes to the way they function or incurring a high integration cost?

The metrics allowed us to highlight the advantages that interpreted runtime models provide for integrating adaptive UI capabilities in existing systems.

After integrating RBUIS in OFBiz, we demonstrated its runtime scalability and efficiency, thereby answering research question Q6 (Chapter 3):

Q6: Does the proposed UI adaptation technique (RBUIS) provide a real-time runtime performance and is it scalable?

We interviewed the manager of a software company, which builds products based on OFBiz, in order to get an industry perspective about our adaptation approach and answer research question Q7 (Chapter 3):

Q7: What is the perspective of industry experts on the generality and flexibility of the devised UI adaptation technique (RBUIS)?

Finally, we conducted two usability studies, one online and another in the laboratory. These studies showed that RBUIS can significantly improve usability when applied to real-life enterprise application scenarios, thereby answering research question Q8 (Chapter 3):

Q8: Does feature-set minimization and layout optimization significantly improve the usability (efficiency, effectiveness, and satisfaction) of enterprise application user interfaces?

7.2 Evaluation Based on Technical Metrics

Many enterprise applications incorporate hundreds or even thousands of UIs and are already at a mature stage in their development. A method is needed for integrating adaptive UI capabilities into these systems, without incurring a high development cost or significantly changing the way they function.

In his paper on criteria for evaluating UI research, Olsen, Jr. (2007) gives an example about the objections that were made in the late 1970s towards new UI architectures due to the large amount of legacy code written for command-line or text UIs. He notes that legacy code can be a barrier to progress hence, if rewriting applications is necessary, it could be the price of progress. Yet, Olsen also states that providing a new advance while maintaining legacy code is desirable. The latter is what we aim to achieve with our method for integrating adaptive UI capabilities in enterprise applications.

Another integration challenge lies in the difference between research work on adaptive user interfaces presented in the literature and traditional UI development techniques. For example, many research works on adaptive UIs adopt the model-driven approach to UI development either partially or fully such as: Supple (Gajos et al. 2010) and MASP (Blumendorf et al. 2010) respectively. However, despite the advantages of the model-driven approach, the user interfaces of many existing software systems including enterprise applications have been developed using traditional techniques. Therefore, an important issue to consider for adaptive UI integration in existing applications is the means of combining new UI development approaches such as the model-driven approach with UIs that have been built using existing UI design tools such as interface builders.

We present a method for integrating adaptive UI capabilities in enterprise applications without the need for major integration effort. We integrated RBUIS in the open-source enterprise application Apache Open For Business (OFBiz). Several technical characteristics, related to RBUIS and the integration method, were evaluated by establishing and applying technical metrics. This evaluation covered: reverse-engineering, integration, and runtime execution.

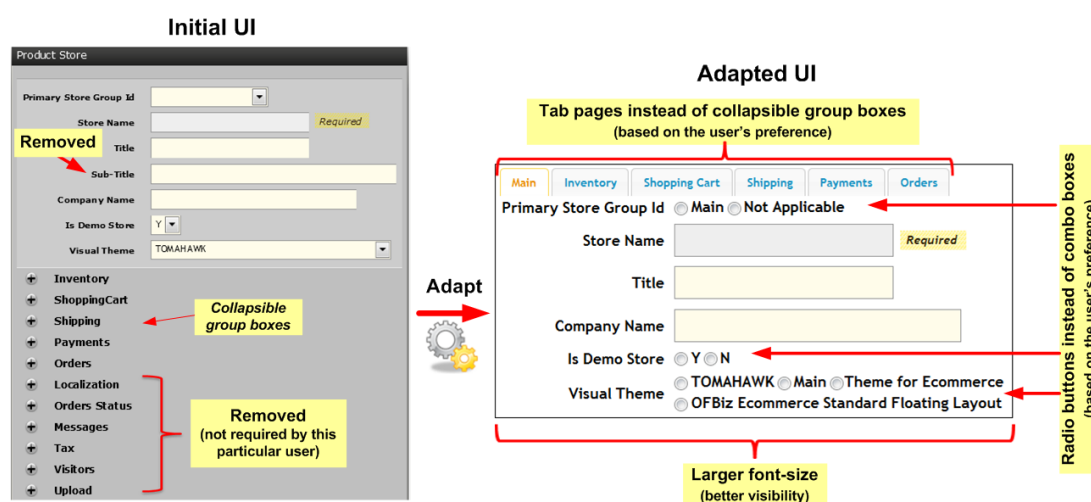


Figure 7.1: An Example on Adapting the Product Store UI of OFBiz

We followed the Cedar Architecture to integrate RBUIS in OFBiz. The example shown in Figure 7.1 was part of the evaluation we conducted. It demonstrates feature-set minimization and layout optimization operations on the *Product Store* UI of OFBiz.

7.2.1 How Does the State-of-the-Art Integrate in Existing Systems?

We established in Chapter 2 that the existing art does not provide means for integrating adaptive UI capabilities in existing system. This section groups the state-of-the-art according to the adaptation approaches, and discusses the strengths and shortcomings of each approach in terms of integrating into existing systems.

Toolkit-based approaches for adaptive UIs have been explored extensively in the literature. Some examples include: caring, sharing widgets (Lecolinet 2003), selectors (Johnson 1992), SwingStates (Appert & Beaudouin-Lafon 2006), etc. Technology dependence is one of the disadvantages of toolkits in comparison to model-driven UIs. This disadvantage could impact the integration of adaptive UI toolkits in existing enterprise applications since the entire toolkit has to be redeveloped for each technology. Providing technology-independence is an important part of the Cedar Architecture. The Comet(s) (Calvary et al. 2005) attempts to combine the toolkit and model-driven approaches for building adaptive UIs. Nevertheless, even if the toolkit was

technologically compatible with an existing enterprise system, the amount of code modification that is required to switch the UI from the classical toolkit to the adaptive one could be significant. This is especially true if the enterprise application's UI was not developed by following design patterns like a “bridge” to decouple each widget's abstraction from its implementation. In such a situation, a conversion tool is necessary with some manual work for shifting the UI specification from one toolkit to another. Our approach operates on existing UIs without updating them to a new toolkit due to the loose coupling between the adaptation mechanism and the technology dependent UI representation.

Aspect-oriented programming (AOP) was proposed for improving the separation of concerns in software systems (Kiczales et al. 1997). One approach that used AOP for adapting UIs, requires several presentations to be defined for the same UI at design-time, and uses a weaver to associate these presentations to instrument classes that handle the way the UI functions (Blouin et al. 2011). Our approach is conceptually similar to AOP since we are trying to achieve a separation of concerns between the UI adaptation technique and the enterprise system. Yet, our main focus is on adapting the UI's presentation and not its implementation code. From this perspective, the existing AOP-based approach requires UI variations to be defined manually by developers at design-time, whereas our approach aims at adapting UIs through adaptive behavior using rules that could be applied to different UIs at runtime. For example, adaptive behavior could be defined to switch the way the UI's widgets are grouped by changing group-boxes to tab-pages. Adaptive behavior defined outside the enterprise system could save integration time and support dynamic changes that narrow the gap between development-time and runtime.

Design-time model-driven approaches rely on ***generating*** multiple adapted UIs based on models that represent the UI at several levels of abstraction. Some approaches rely on software-product-lines (SPL) for tailoring user interfaces. Although SPLs can be dynamic (Bencomo et al. 2008), SPL-based UI adaptation approaches, such as the work by Pleuss et al. (2010), focus on design-time adaptation like generating UIs with different subsets of features based on a feature model, whereas runtime adaptive behavior is not addressed. Smart templates are another generative approach and were used with

ubiquitous remote control mobile UIs (Nichols et al. 2004). Code generation makes such approaches difficult to adopt for existing mature enterprise applications due to the amount of effort needed to integrate the generated code in the existing systems and the increased number of software artifacts that can require maintenance. Also, if the adopted presentation technology required compilation, for example Windows Forms, adding UI artifacts would increase the compilation time. Our integration method requires a few lines-of-code to be added to the enterprise application at design-time to trigger UI adaptations at runtime. Therefore, RBUIS can be integrated in existing systems without major design-time effort or the need for creating or changing a large number of software artifacts.

Runtime model-driven approaches keep the models alive at runtime for adapting the running UI dynamically. Some are generative, such as MASP (Blumendorf et al. 2010), thereby generate an individual UI specification from the models at design-time and use the models to adapt this UI at runtime. Other approaches such as: Supple (Gajos et al. 2010) and DynaMo-AID (Clerckx et al. 2005), rely on **interpreting** the models and dynamically rendering the UI. MASP, Supple, and DynaMo-AID did not demonstrate and evaluate the ability to integrate their proposed approaches in existing software systems.

We think that runtime model-driven UI development is the most suitable approach to support a method for integrating adaptive UI capabilities in existing enterprise applications, due its dynamic nature. Yet, the lack of attention from existing works in the literature towards integration drives us to present an integration method based on the Cedar Architecture. This method allows us to integrate RBUIS in OFBiz and to conduct a metric-based evaluation.

7.2.2 Integrating RBUIS in OFBiz

This section provides an overview of our method for integrating adaptive UI capabilities (RBUIS) in enterprise systems based on the Cedar Architecture. The open-source enterprise application OFBiz is used as a test-case.

OFBiz¹¹ is an open-source enterprise automation software project, which contains several sub-systems such as: enterprise resource planning (ERP), manufacturing resources planning (MRP), customer relationship management (CRM), e-business and e-commerce, and supply chain management (SCM). It can be considered as a general-purpose, large-scale, enterprise system considering the characteristics shown in Table 7.1.

Table 7.1: Some of the Characteristics of OFBiz

OFBiz Release 12.04	
Number of User Interfaces	> 750
Number of Lines-of-Code	≈ 1,466,000
Projects Based on OFBiz	20
Public Sites using OFBiz	90

Commercial enterprise systems can be larger. For example, SAP has over 250,000,000 lines-of-code (Judith Hurwitz 2007) and Lawson has over 10,000 UIs (Allbee 2008). Nonetheless, OFBiz has complex UIs with a large number of widgets, which may need adaptation, making it a good candidate for our study. For example, the main UIs from its Catalog module have an average of 55 widgets and a maximum of 170. Also, an open-source system is necessary to test our integration method. Our method could work with commercial systems but the company that owns the source-code should perform the integration.

7.2.2.1 Integration Based on the Cedar Architecture

The Cedar Architecture's (Figure 4.1 in Chapter 4) client components can integrate in enterprise applications to empower them with adaptive user interface capabilities as illustrated by arrows (1) to (5) in Figure 7.2, using OFBiz as an example.

¹¹ Apache Open For Business: <http://ofbiz.apache.org>

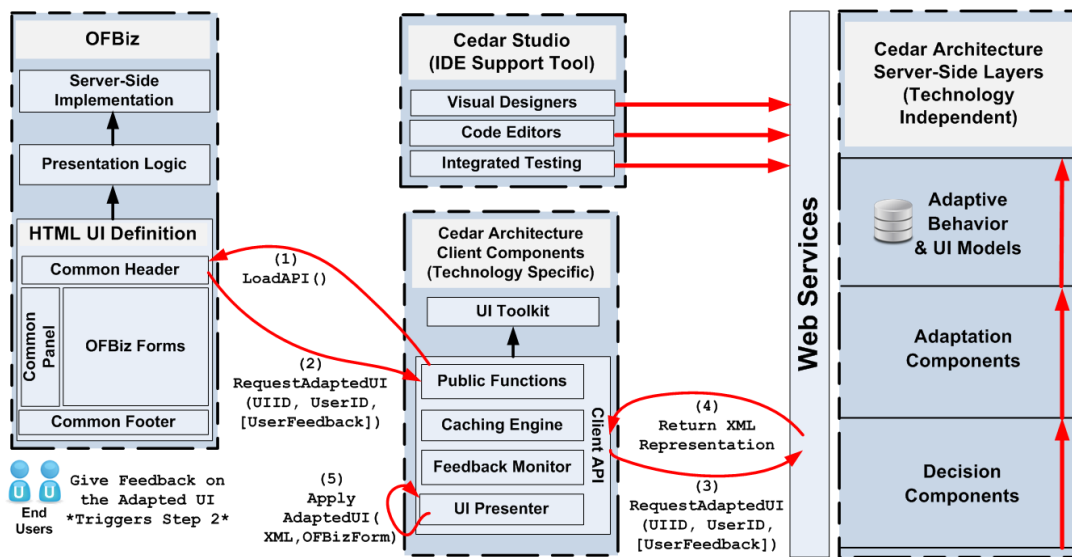


Figure 7.2: Integrating Adaptive User Interface Capabilities (RBUIs) in OFBiz based on the Cedar Architecture

These components are dependent on the programming and presentation technologies, since they have to be integrated in the enterprise application's code. Hence, different sets of components are required. These components offer an application programming interface (API) that is loaded globally (1) in the enterprise application, for example in OFBiz's common header. Whenever the end-user launches a UI, a request is made to the API for adapting this UI; the identifiers of the end-user and the UI are passed as parameters (2). The API uses web-services to pass the UI adaptation request to the server-side layers (3), which perform the adaptation and return the result to the API as XML (4). The API's UI Presenter is responsible for applying the adaptation result to the running enterprise application UI, which is an HTML page in the case of OFBiz. Once a UI is adapted, the Caching Engine will cache the adapted version on the client-side in case the end-user requests it again.

7.2.2.2 The Technique for Integrating RBUIs in OFBiz

OFBiz uses HTML to represent its UIs. Hence, in order to integrate RBUIs in it, we developed a JavaScript version of the Cedar Architecture's client API, which works with HTML UIs. Since RBUIs adopts a model-driven UI development approach, we devised a procedure for reverse-engineering HTML forms into a model-driven representation supporting the levels of abstraction

suggested by CAMELEON (task, AUI, and CUI models). The reverse-engineering is done at design-time. However, our technique launches the HTML pages of OFBiz in the browser then acquires the HTML through JavaScript to include the elements that are generated by server-side scripts. Our procedure transforms an HTML form into an XML document, which is used to create a CUI model. Then, the CUI is reverse-engineered into an AUI model and the AUI into a task model automatically. The only manual part in this procedure is the definition of mapping rules (Figure 6.6 and Figure 6.8), which control how models are mapped to a higher level of abstraction.

**Listing 7.1: Code for Reverse Engineering HTML UI to a Model-Driven Representation:
Excerpt of HTML Table Example**

```
1.  function ConvertHTMLTableToXml(TableID) {
2.    var xml = "";
3.    $("#" + TableID + " tr").each(function () {
4.      //Parse Cells
5.      var cells = $("td", this);
6.      for(var cellCtr=0; cellCtr<cells.length; ++cellCtr) {
7.        var inputs = $("input", cells.eq(cellCtr));
8.        //Parse Input Fields
9.        for(var inpCtr=0; inpCtr<inputs.length; ++inpCtr) {
10.         var fieldType = inputs.eq(inpCtr).attr('type'),
11.         fieldID = GetFieldID(inputs.eq(inpCounter)),
12.         element = GetElement(fieldID);
13.         //Generate XML for Element
14.         var xmlInput = GetInputFieldXml(
15.           element, fieldType, fieldID) + "\n";
16.         xml += xmlInput;
17.       }
18.     }
19.     return xml;
20. }
```

The process of transforming the HTML FUI into a CUI model is done using JavaScript. The excerpt of the code for reverse-engineering an HTML table is shown in Listing 7.1. Mapping the CUI to an AUI and the AUI to a task model can be done visually using Cedar Studio as shown in Chapter 6.

After reverse-engineering the UIs that require adaptation, we can apply RBUIS on the obtained UI models using Cedar Studio. To make the adaptation work at runtime on OFBiz's HTML pages, we need to extend OFBiz with a few lines-of-code that load the client API, call the web-service, and apply the obtained result. OFBiz uses a master page to wrap its UI forms with a common header, footer, and panel as shown in Figure 7.2. To reduce the integration effort we loaded the API and performed the adaptation call in the common header using the code shown in Listing 7.2.

Listing 7.2: Code for Enabling Adaptive UI Capabilities

```
1. //Load the API Scripts
2. <script type="text/javascript" src="http://
   [ServiceAddress]/CedarScripts.js"></script>
3. <script type="text/javascript">
4.     $(document).ready(function() {
5.         //Setup the API
6.         Initialize('[ServiceAddress]');
7.         //Call the API to adapt the UI and
           //pass the logged-in user id as a parameter)
8.         LoadAdaptedUI(getUserID());
9.     });
10. </script>
```

The “`getUserID()`” function call on Line 8 in Listing 7.2 should be implemented by a developer to obtain the identifier of the logged-in user from the OFBiz system. The “`LoadAdaptedUI`” function can internally acquire the UI identifier through a mapping table, which contains the UI's URL and a number to identify the UI's models in the database hosted on the Adaptive Behavior and UI Models layer. The UI's URL is obtained from the web-browser and passed as

a parameter to the adaptation function on the web-service. The mapping is done on the server-side by querying a mapping table in the database.

After receiving an XML representation of the adapted UI from the server, the UI presenter component will apply the changes to the HTML page loaded on the client machine by modifying the widgets' properties. An excerpt of the code that applies the adaptations is shown in Listing 7.3. This code excerpt demonstrates hiding the widgets that were set to be invisible by an adaptation, for example by removing features that are not required by a certain end-user.

Listing 7.3: API Code for Applying the Adapted UI: Excerpt of Widget Hiding Example

```
1.  function ApplyAdaptedUI(UIXML) {  
2.      //Loop around the UI widgets  
3.      $(UIXML).find("Control").each(function () {  
4.          //Get the name and visibility attributes  
5.          var technicalName = $(this).attr('TechnicalName');  
6.          var isVisible = $(this).attr('Visible');  
7.          //Hide the invisible elements  
8.          if(isVisible == 'false') {  
9.              var element = GetElement(technicalName);  
10.             //Hide the element if it exists  
11.             if (typeof (element) != 'undefined')  
12.                 { element.style.visibility = 'collapse'; }  
13.         }  
14.     });  
15. }
```

7.2.3 Metric-Based Evaluation

The process of integrating RBUIS in enterprise applications starts by reverse-engineering the target application's UIs. Afterwards, the application is extended to support adaptation hence becoming able to adapt its UIs at runtime. This section explains the metrics, which we used to evaluate our integration method at all the stages of the process and demonstrates an application of these metrics to scenarios from OFBiz.

7.2.3.1 Reverse-Engineering the User Interfaces

As we mentioned in Section 7.2.2.2, we devised a procedure for reverse-engineering HTML forms into a model-driven representation that can be adapted by RBUIS. Although it is automated, this procedure requires mapping rules to be defined manually. Hence, the first question that might come to mind is about the difficulty of deducing these rules from the existing enterprise system since it has a large number of UIs. Assuming that there is no prior knowledge of the types of mapping rules required for reverse-engineering the enterprise system at hand, we defined the following metrics for estimating the number of UIs that require manual work before the majority of the mapping rules are detected. These metrics indirectly show the level of diversity in an application's UIs. More diversity could signify that there are more mapping rules, which are more uniformly distributed over the entire system.

The **approximate mapping rule detection saturation point** SP indicates that the number of new encountered mapping rules stabilized after reverse-engineering a number of UIs a . This metric will allow us to test if the Pareto principle (70-30 rule) applies for detecting 70% of the mapping rules in the first 30% of the UIs. This metric gives an indication about the similarity among the UIs of a software system. Although eventually all the mapping rules have to be determined, the applicability of the Pareto principle indicates that the manual work is concentrated at the beginning of the reverse-engineering process. Therefore, experienced software developers could work on the first 30% of the UI and leave the more straightforward work (70%) for the developers with less experience. The developers who have less experience can trigger the automated process and test the UIs. In case there is a problem with a UI due to the need for a new mapping rule, the experienced developers can step-in to define this rule. If the Pareto principle stands, there could be a better allocation of human resources by combining automation with minimal human control from junior developers in 70% of the reverse-engineering process.

To check if the **Pareto principle** holds, we define the following equation where $\{R\}$ is the set of rules detected in the UIs before SP and $\{MR\}$ is the set of all the detected mapping rules:

$$P = \frac{|\{R\}|}{|\{MR\}|} : SP(\{MR\}) \leq 0.3$$

Equation 7.1: Pareto Principle for Mapping Rule Detection Metric

The saturation point SP is defined as follows:

$$SP(\{MR\}) = \frac{UI_a}{T} \mid \forall UI_b \mid b > a : C_b = C_{b+1} \pm \varepsilon$$

Equation 7.2: Approximate Mapping Rule Detection Saturation Point Metric

where UI is a user interface being reverse-engineered, C is the number of new mapping rules detected in this UI, the subscript b of C indicates the next UI to be reverse-engineered, and T is the total number of UIs to be reverse-engineered. The types of mapping rules that are encountered when reverse-engineering a UI can differ depending on the characteristics of the software application being reverse-engineered. We hypothesize that the Pareto principle holds for enterprise applications due to the use of similar WIMP style UIs.

OFBiz Scenario: We selected a sample formed of the 19 main input UIs from the “Catalog” and “Human Resources” modules. We were able to deduce two types of mapping rules necessary for reverse-engineering these UIs into a model-driven representation: (1) The most common type of rule is the one that maps individual HTML elements to CUI elements, which are in turn mapped to AUI elements then tasks in the task model, and (2) the second type of rule is related to grouping widget pairs composed of a label and an input widget into logical groups, which are reflected in the AUI and task models. The mapping rules are used for linking the elements between one UI level of abstraction and another. These rules can be defined using Cedar Studio as shown in Chapter 6 by Figure 6.6 and Figure 6.8. Defining rules from these two types alongside getting information provided by the HTML UI (e.g., widget properties: name, size, location, etc.) was sufficient to obtain a model-driven UI representation that we can adapt using RBUIS. In case the software application at hand required more advanced mappings rules, a transformation language such as XSLT could be introduced into Cedar Studio for supporting this functionality.

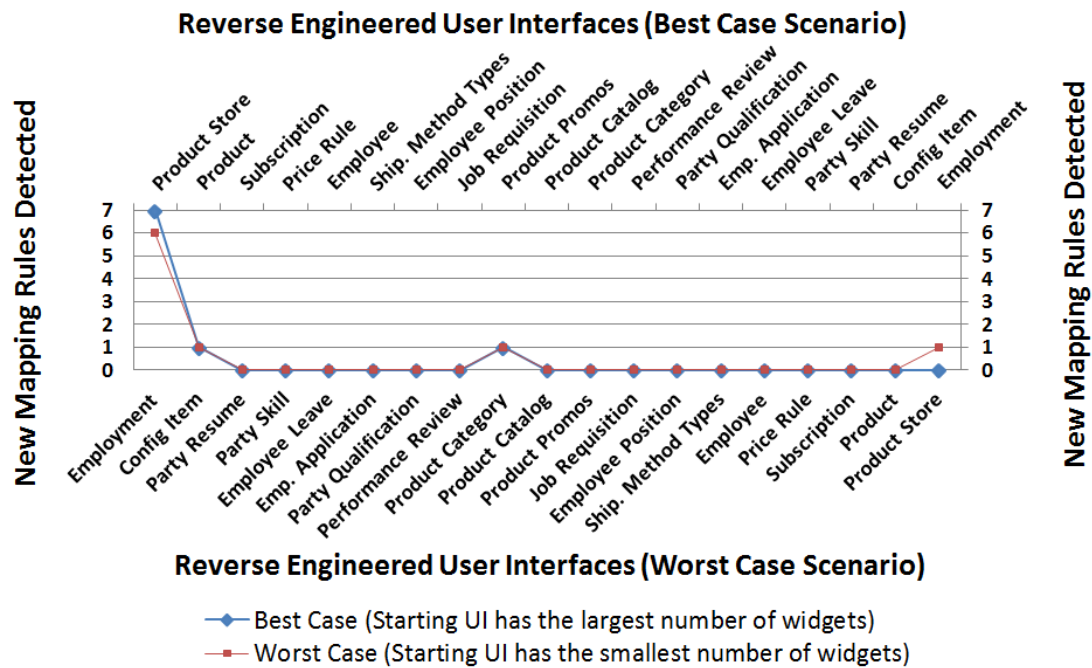


Figure 7.3: Saturation Point for Mapping Rules

The chart in Figure 7.3 shows the number of mapping rules, which we deduced from each of the OFBiz UIs that we selected. The chart shows a best case scenario where we started with the UI that has the largest number of widgets and a worst case scenario where we did the opposite. We encountered 8 different widget types, each requiring 1 mapping rule, and were able to detect the second mapping rule relating to logical widget grouping in the first UI. We obtained a saturation point $SP = 2 / 19 = 0.10$ signifying that after the second UI the mapping rules become minimal as shown in Figure 7.3. Following our example where $SP = 0.1$, P is: $7 / 9 = 0.77$ (77%) in the best case scenario and $6 / 9 = 0.66$ (66%) in the worst case one. With an average of 71.5% of the rules detected in the first 10% of the UIs, we can say that the Pareto principle holds and the UIs of OFBiz are highly similar.

7.2.3.2 Integrating the Adaptive UI Capabilities

After reverse-engineering the UIs, we can assess the level of change the integration will incur on the enterprise application. We defined the *lines-of-code* and *change-impact* metrics for this assessment.

The **lines-of-code** metric refers to the code required locally in each UI or globally in the enterprise application to apply a type of adaptation. This metric excludes the API code since the Cedar Architecture requires each presentation technology (e.g., HTML) to have one API that is reusable with any enterprise application. The lines-of-code metric is given as follows:

$$LLOC(A, UI) \in N, \quad GLOC(A, EA) \in N$$

Equation 7.3: Lines-of-Code Metric

where *LLOC* represents a UI's local lines-of-code, whereas *GLOC* represents the global lines-of-code common across the application, *A* is the required adaptation, *UI* is the user interface to which this adaptation will be applied, and *EA* is the enterprise application. The values for *LLOC* and *GLOC* represent the number of lines-of-code that must to be added to make the adaptation operational.

OFBiz Scenario: As an example test-case, we considered the context-driven UI adaptations listed in Table 7.2 and applied them to OFBiz. An example of the output was shown earlier in Figure 7.1. Adaptation A1 is a feature-set minimization, whereas adaptations A2, A3, and A4 are layout optimizations.

Table 7.2: Example User Interface Adaptation Operations

Code	Adaptation
A1	Reduce features (e.g., hide or disable widgets)
A2	Switch widget type (e.g., combo-boxes to radio-buttons)
A3	Change layout grouping (e.g., group-boxes to tab-pages)
A4	Change font-size (e.g., larger fonts for visually impaired users)

Our method only requires the 10 lines-of-code shown in Listing 7.2 to be added globally to OFBiz's common header to empower it with adaptive UI capabilities. Consider {AE} to be the set of adaptations listed in Table 7.2. The lines-of-code needed to make these adaptations work in OFBiz using our method are $\forall x \mid x \in \{AE\}, GLOC(x, OFBiz) = 10$ and $LLOC(x, AnyUI) = 0$. Achieving this low number of lines-of-code is possible because all the adaptation rules are defined on the server-side as shown in Figure 7.2.

Some approaches discussed in Section 7.2.1 operate by changing the UI's representation (e.g., HTML tags) at design-time. Therefore, we established the **change-impact** (CI) metric to measure the level of change each approach will incur on the enterprise application. A higher change-impact could signify that: (1) More time and effort are needed to perform the integration and (2) the compilation time will increase if a compiled presentation technology such as Windows Forms was used. Since we can think of UIs in terms of widgets, the change-impact metric is given as follows:

$$CI(A, UI) = \sum_{k=1}^n l_k \times |\Delta\{W\}_k| \times v$$

Equation 7.4: Change Impact Metric

where A is the adaptation being applied, UI is the user interface being adapted, k is a type of widget (e.g., text-box, combo-box, etc.), n is the number of widget types in the UI , l_k is the number of lines required for representing each widget type (e.g., number of HTML tags), and $|\Delta\{W\}_k|$ is the number of widgets of a certain type that have been changed by the adaptation.

The variable v represents the number of generated UI versions and is > 1 for approaches that cannot adapt the same UI copy (e.g., a single HTML page) but generate multiple copies of the UI, each of which is adapted to a certain context-of-use. Widget toolkits aim at replacing existing widgets from the standard toolkit with adaptive equivalents. Hence, the value of v for widget toolkits would be $= 1$ since the change is occurring in the initial UI copy. We should note that widget toolkits are generally used to adapt the layout and do not have the ability to adapt the feature-set due to their lack of a high-level UI model such as a task model. Model-driven design-time generative approaches generate multiple versions of the same UI adapted to different contexts-of-use. Hence, the value for v in this approach would be > 1 . The research work that used AOP for adapting the UI's behavior (Blouin et al. 2011), relied on manually creating multiple adapted UI layouts hence we also consider its v value to be > 1 . As for our method, CI is always $= 0$ since we use runtime adaptation hence the UI representation (e.g., HTML pages) will remain completely intact at design-time.

Table 7.3: Integration Time of Different Adaptation Approaches

Approach	Integration Time
Runtime Interpreted Model-Driven	Low
Design-Time Generative Model-Driven	Medium
Widget Toolkits	Medium / High
AOP + Design-Time Manual Adaptation	High

Based on CI we provided a qualitative comparison between the different UI adaptation approaches as shown in Table 7.3. Our aim is to give an idea about the differences in the required integration effort between approaches, while recognizing that there could be slight differences between adaptation techniques using the same approach. Widget toolkits require an average amount of time if a conversion tool existed to automatically convert the UI; otherwise a high amount of time is needed. Model-driven generative design-time approaches require an average amount of time since the adapted versions could be automatically generated but more time could be still required to integrate them with the software application. Logically, manual adaptation requires a high amount of time. The integration time of our method is low since CI is always = 0, hence the developers can continue working on the application without major disruptions.

OFBiz Scenario: We attempted to apply adaptation A2 (Table 7.2) to the 19 main input UIs of the Catalog and Human Resources modules of OFBiz. This adaptation switches combo-boxes with three other types of widgets including: radio-buttons, list-boxes, and lookups. These possibilities indicate that we could obtain three different versions of the UI, hence v (Equation 7.4) = 3 for the model-driven generative and manual design-time approaches and $v = 1$ for the widget toolkit approach. The value for n (Equation 7.4) is 1 since we are only adapting combo-boxes, and we consider that each combo-box is represented by a single HTML tag hence l (Equation 7.4) = 1. The results we obtained from calculating CI are listed in Table 7.4, and show that our approach, namely runtime interpreted model-driven, has the lowest change-impact. The adaptation we applied in this example switches combo-boxes with radio-buttons, list-boxes, and lookups.

Table 7.4: Results Obtained from Calculating the Change Impact Metric

Approach	Change-Impact	
	Mean	Total
Runtime Interpreted Model-Driven	0	0
Widget Toolkits	6.94	132
Design-Time Generative Model-Driven	106.73	2028
AOP + Design-Time Manual Adaptation	106.73	2028

7.2.3.3 Level of Decoupling

The level of decoupling shows how much intertwining exists between the adaptive behavior and the enterprise application. It is affected by the percentage of adaptive behavior defined in the enterprise application versus that defined separately. Decoupling provides a separation of concerns that could offer potential for scalability and facilitate the integration of an adaptation technique in existing enterprise applications. As shown earlier in Figure 7.2, the Cedar Architecture completely separates the implementation of the adaptive UI technique, which resides on a server and the enterprise application that uses a client-side API to communicate with it through a web-service.

It is important to maintain the **backward compatibility** of UI adaptations as enterprise applications evolve. We consider an adaptation *A* to be backward compatible if it can be applied to previous UI versions *successfully* and *without reintegration effort*. Decoupling helps in improving backward compatibility in terms of eliminating *reintegration effort*. A conceptual assessment of the backward compatibility of UI adaptation approaches is presented in Table 7.5 based on the need for *reintegration effort*.

Table 7.5: Backward Compatibility of UI Adaptation Approaches

Approach	Backward Compatible
Runtime Interpreted Model-Driven	True
Widget Toolkits	Depends on the ability to load a new widget toolkit version at runtime
Design-Time Generative Model-Driven	False
AOP + Design-Time Manual Adaptation	False

Widget toolkits can be backward compatible if it is possible to load a new toolkit version at runtime to update the existing adaptive behavior in older versions of the enterprise application. This is not possible with model-driven approaches that generate UIs at design-time since the generated artifacts have to be manually integrated in all the previous versions. Manual design-time adaptation suffers from a similar problem. If we consider the adaptations listed in Table 7.2, we can say that our approach is backward compatible since it is only necessary to define a global code once to make these adaptations work for all the UIs. Hence, the adaptations would work for all the previous versions that have this code since the adaptive behavior is being defined separately.

An adaptation's *success* can be partially due to differences in the UI definition between one version and another. We defined a metric for calculating the backward compatibility success ratio as follows:

$$BC(A) = \frac{|\{AW(UI_{vn})\} \cap \{AW(UI_{vn-k})\}|}{|\{AW(UI_{vn})\} \cap \{W(UI_{vn-k})\}|}$$

Equation 7.5: Backward Compatibility Metric

where UI_{vn} is a UI from the enterprise application version into which the adaptation A was integrated for the first time, and UI_{vn-k} is one of the previous versions; $\{W\}$ is the set of widgets in a UI and $\{AW\}$ is the set of widgets affected by an adaptation A .

As an example of partial UI adaptation success, let us consider a UI for managing customer records. Consider that CustomerUI_{v2} has multiple fields, 10 of which are for data selection and are represented as combo-boxes (e.g., gender). Assume that the previous UI version CustomerUI_{v1} has the same data selection fields but only 8 are represented as combo-boxes and the other 2 are list-boxes. If we introduce an adaptation to switch data selection widgets with radio-buttons in CustomerUI_{v2}, we might ignore list-boxes. In this case, $BC = 8/10 = 0.8$ indicating an 80% success rate. With approaches that are not dynamic and rule-based (e.g., design-time generative), two adapted UIs have to be generated and integrated into each respective CustomerUI version to achieve a 100% success rate. As for our approach, we only have to adjust the adaptation

rule in our RBUIS mechanism to take into consideration list-boxes as well as combo-boxes to obtain a 100% backward compatibility.

7.2.3.4 Runtime Performance

Considering that our approach is highly dynamic, we had to test its runtime *efficiency* and *scalability* especially since we are working with UIs that are expected to load in real-time. In Chapter 5, we conducted a complexity analysis to show that the algorithms behind our RBUIS mechanism are theoretically scalable. In this chapter, we demonstrate our technique's runtime efficiency and scalability after integrating it in an existing real-life system (OFBiz). To perform this test, we defined the following **efficiency** metric as a function of an adaptation A and a user interface UI :

$$E(A, UI) = t_{0a} + t_{0b} + t_1 + t_2$$

Equation 7.6: Runtime Efficiency Metric for UI Adaptation

where t_{0a} is the time needed to perform an adaptation on the server-side, t_{0b} is the common server-side time needed for any number of adaptations, for example loading common data before applying the adaptations, t_1 is the time needed to transmit the adapted UI as XML to the client, and t_2 is the time it takes the API to apply the adaptation on a running UI such as an HTML page in OFBiz.

We used this metric to test the efficiency of the four example adaptations listed in Table 7.2 on the three UIs with the highest number of widgets in OFBiz's Catalog module. The test was conducted on a single machine with an Intel Core 2 Duo 2.93 GHz CPU and 4 GB of RAM running a 32 bit edition of Windows 7. We used the Firefox web-browser to run OFBiz.

We determined the t_{0b} variable to be equal to 30 milliseconds (ms). The t_1 variable depends on the network connection and is negligible for our test since we were operating on a single machine. We calculated the average XML document size for the 3 selected UIs to be 20kb. Based on this file size, t_1 will be very small over an internet connection (e.g., $\approx 15\text{ms} / 10\text{Mbps}$) and negligible

over a corporate network (e.g., $\approx 0.15\text{ms} / 1\text{Gbps}$). The values of variables t_{0a} and t_2 are shown in Figure 7.4 for each UI and adaptation.

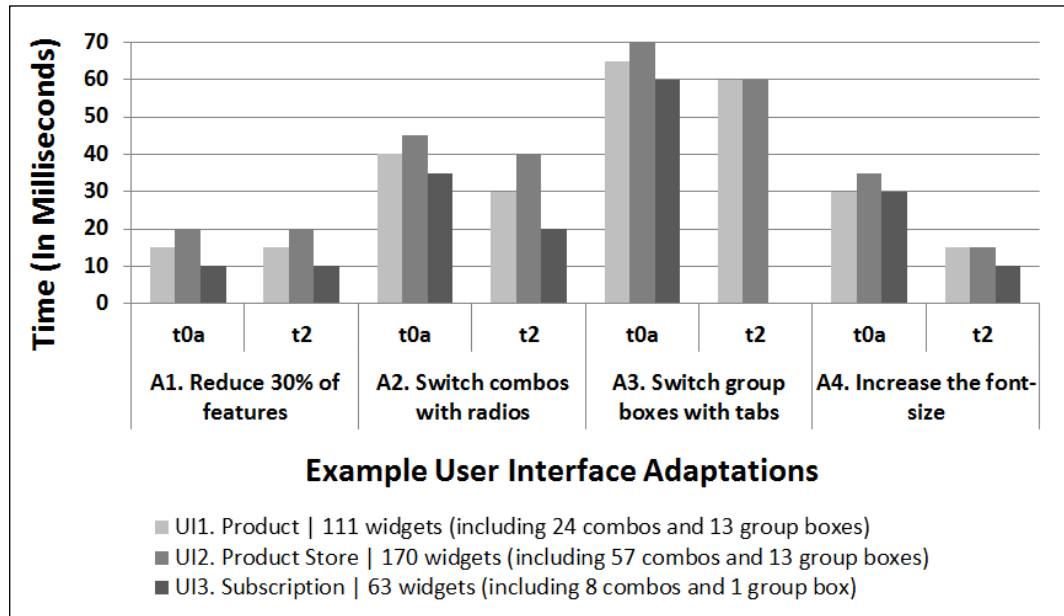


Figure 7.4: Results of the Efficiency Test on Three OFBiz UIs Using Four Example Adaptations ($t_{0b} = 30\text{ms}$ and $t_1 = 15\text{ms}$)

Using the data shown in Figure 7.4 and considering t_1 to be 15ms, we determined the average efficiency for each adaptation to be: $E(A1) = 75\text{ms}$, $E(A2) = 115\text{ms}$, $E(A3) = 150\text{ms}$, and $E(A4) = 90\text{ms}$. The general average is $(75 + 115 + 150 + 90) / 4 = 107.5\text{ms}$. If we do not consider the fixed values t_{0b} (30ms) and t_1 (15ms), the general average will be 62.5ms. Based on this number, we can say that our technique can perform around 15 different adaptations on the same UI, transmit it, and display the result all in less than 1 second ($62.5 \times 15 + 30 + 15 = 982.5\text{ms}$).

Since the Cedar Architecture supports client/server-side caching, performance can be further enhanced. Client-side caching is used if an end-user who is still operating in the same context, for example still logged-in with the same roles, requests a UI that has already been adapted. In this case the efficiency metric will be: $E(A, UI) = t_2$ (general average 24.5ms). As for the server-side caching, it is used when an end-user requests a UI that has already been adapted for another end-user operating in the same context, for example someone who has the same roles. In this case, the efficiency metric will be: $E(A, UI) = t_1 + t_2$.

After testing the efficiency of our technique we verified its **scalability** by load-testing the UI adaptation web-service. We selected the largest of the three UIs that were used in the scalability test, namely the *Product Store* UI with 170 widgets, and applied to it the four adaptation operations shown in Table 7.2. We submitted increasing requests of that UI to the server over five minute periods and repeated the whole cycle five times. The web-service was hosted on an Amazon cloud server with a single Intel Xeon CPU with 2 cores (2.40 GHz, 2.15GHz), 3.75 GB of RAM, and running a 64-bit edition of Windows Server 2012 Standard with the IIS 7 web-server. We consider this setup to be an average configuration since enterprises with hundreds of users usually setup servers with multiple CPUs and a larger amount of RAM. We simulated the load using an application that we developed and ran simultaneously on three client machines. The resulting server response times ($t_{0a} + t_{0b}$ from Equation 7.6) are shown as a box-plot in Figure 7.5.

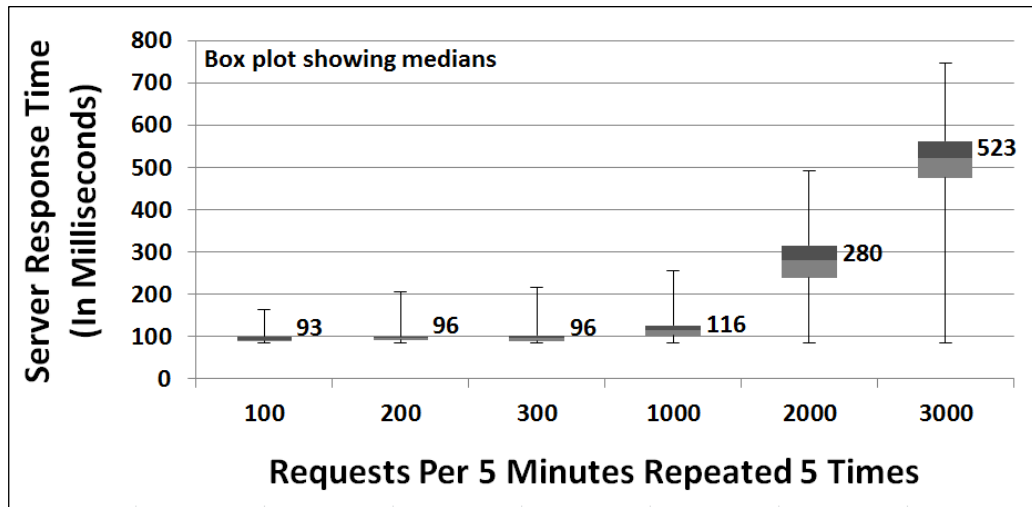


Figure 7.5: Box-plot of Load-Testing Results (showing medians)

The fitting curve of the mean response times shown in Figure 7.6 is polynomial of the 4th order with $R^2=0.9999431$. We should note that the polynomial curves of the 2nd and 3rd orders also produced a high R^2 where R^2 (2nd) = 0.9977252 and R^2 (3rd) = 0.9989506. Based on this test, we can say that our UI adaptation service is scalable and will not form a bottle-neck if it receives a high number of requests.

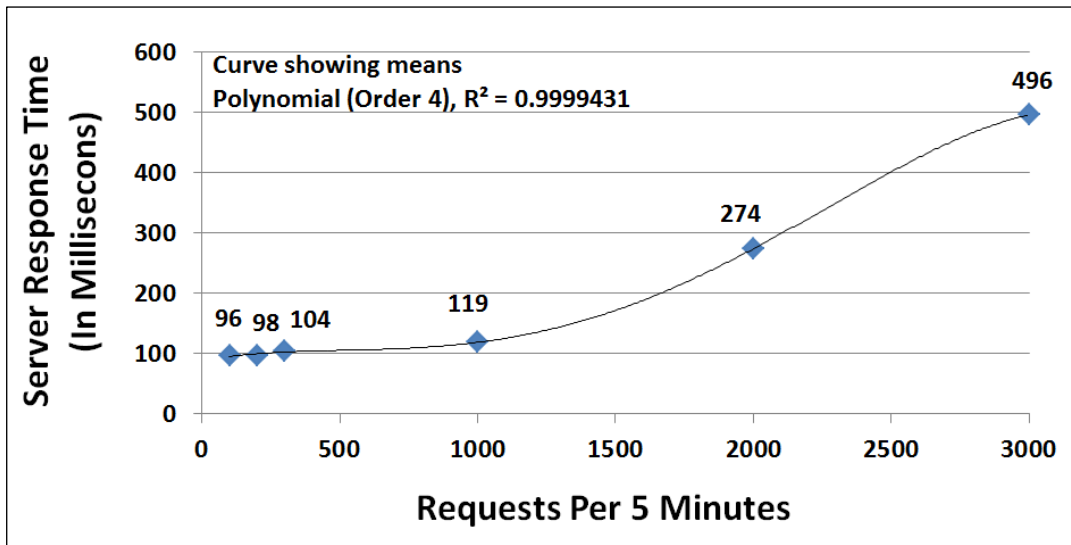


Figure 7.6: Curve of the Load-Testing Results (showing means)

7.2.4 Discussing the Results of the Technical Evaluation

This section discusses the results of the technical evaluation in terms of supporting or rejecting the null hypotheses H_{0-1} and H_{0-2} that we established in Chapter 3. The limitations and threats to validity are also presented.

7.2.4.1 Discussion

Reverse-engineering the UIs of two modules from OFBiz showed that its UIs are highly similar and require a few mapping rules to be reverse-engineered from HTML into a model-driven representation. This similarity makes the reverse-engineering process easier for enterprise applications, which contain thousands of user interfaces.

The saturation point metric that we defined in Section 7.2.3.1 showed that 71.5% of the mapping rules were determined in the first 10% of the UIs. This distribution allows a better allocation of human resources by combining automation with minimal human control from junior developers in over 70% of the reverse-engineering process. Hence, the senior developers might only have to intervene when the definition of new mapping rules are needed.

We were able to integrate RBUIS by merely adding a few lines-of-code globally in the OFBiz. Hence, our integration approach causes less change to

the enterprise application than other existing approaches as shown by the change-impact metric in Section 7.2.3.2. Our adaptation approach also provides better backward compatibility (Section 7.2.3.3) when applying adaptations to older UI versions, due to the definition of the adaptive behavior separately.

Based on the abovementioned results, we reject the null hypothesis H_{0-1} because we showed that using the Cedar Architecture and interpreted runtime models allowed RBUIS to integrate in OFBiz without causing major changes to the way it functions or incurring a high integration cost.

We also demonstrated in Section 7.2.3.4 that our technique can perform around 15 different adaptations on the same UI, transmit it to the client machine, and display the result all in less than 1 second. Therefore, we can reject the null hypothesis H_{0-2} because our UI adaptation technique provides real-time performance (milliseconds) and is scalable.

7.2.4.2 Threats to Validity and Limitations

The data presented in this section is based on applying our UI adaptation approach to scenarios from OFBiz. The figures we obtained by applying the saturation point (SP) metric give us an indication about the nature of enterprise application UIs without claiming generalizability to all enterprise applications. When we compared our approach to others from the literature using the change-impact (CI) and backward compatibility (BC) metrics, we aimed at giving a general conceptual idea about the differences while acknowledging that there could be some variations between the low-level adaptation techniques using the same approach. The load-testing curve presented in Figure 7.6 is intended to show that our UI adaptation mechanism is scalable. Determining an accurate regression equation, which is not the purpose of this test, requires a larger sample of mean execution times.

Task models represented as ConcurTaskTrees support temporal operators, which can help in determining inter-task dependency. Determining this dependency is helpful for feature-reduction adaptation operations. Currently, we are unable to automatically detect these operators when reverse-engineering a UI specified in a presentation technology such as HTML to a model-driven

representation. It is possible to specify these operators manually using the task model design-tool in Cedar Studio.

7.3 Evaluation Based on Industrial Expertise and Data

This section presents an evaluation of the generality and flexibility of our method based on industrial expertise and data. To evaluate our method from an industrial perspective, we drew on the expertise and data from real-life projects offered to us by a software company that sells enterprise systems to medium and large enterprises in China. We selected this company due to its expertise in enterprise systems, UI adaptation, and our test-case OFBiz.

7.3.1 Inquiring about our Approach's Generality and Flexibility

We communicated with the manager of the software company in China multiple times, over the phone, in order to explain our UI adaptation approach and assess how well it fits within their line of work. We discovered that one of the major problems faced by this company is usability related. The enterprise applications that they sell suffer from a diminished user experience due to the diverse end-user needs, which make one UI not fit for all end-users. We established through a verbal explanation of our UI adaptation technique that it could be useful with real-life enterprise systems such as OFBiz.

At a later stage, after integrating our UI adaptation technique successfully in OFBiz, we sought to further evaluate its usefulness by assessing its *generality* and *flexibility*. These two criteria were introduced (alongside others) by Olsen, Jr. (2007) for evaluating UI research including architectures such as the Cedar Architecture. According to Olsen, *Generality* evaluates the possibility of using the proposed solutions with different use cases and *flexibility* evaluates “*the possibility of making rapid design changes that can be evaluated by the users*” (p. 255). We demonstrated our UI adaptation and integration techniques to the manager with videos showing running examples of using our IDE Cedar Studio for developing adaptive model-driven UIs and an example of integrating these capabilities in OFBiz. Afterwards, we conducted a semi-structured interview over

the phone with the manager and followed it with several discussions. During the interview we directly asked the interviewee about his opinion on the generality and flexibility of our UI adaptation approach, without restricting the discussion to these two points to also explore any additional insights that he might have.

To achieve **generality**, our method only requires an API for the presentation technology adopted by the target enterprise application. As shown by the Cedar Architecture in Figure 7.2, all the server-side components are technology-independent and can be accessed from a technology-dependent API through web-services. An API for a particular presentation technology can be used with any application adopting this technology by following the integration procedure described in Section 7.2.2. This was deemed acceptable by the manager since we developed an API and demonstrated it in a working example with Cedar Studio.

According to Olsen's definition, **flexibility** is regarded as a development metric that assesses how easy it is for developers to make rapid design-time changes using a tool. It is achieved from this perspective by Cedar Studio, which supports visual-design tools for both UI models and adaptive behavior in addition to integrated testing of the adapted UIs. These features allow changes and testing to be done rapidly. Based on the videos he observed, the manager thought that Cedar Studio is very promising especially since it supports visual-design tools and the ability to generate one UI model from another.

Although he gave us positive comments on our approach's generality and flexibility, the interviewee did not elaborate a lot on these two points because he was mostly interested in RBUIS's feedback mechanism. Based on his experience, he thought that this mechanism offered flexibility from an end-user perspective. He appreciated the fact that it allows end-users to report their feedback immediately to the system without having to refer to the software company. His comments on this point allowed us to explore an interesting potential for the feedback loop, which we had not previously anticipated. Section 7.3.2 explains this potential and highlights its importance using data from real-life projects.

7.3.2 Importance of the Feedback Loop in the UI Adaptation Process

Based on his company's experience, the manager said that UIs are first adapted by the developers based on the initial knowledge they have on the

needs of an enterprise's end-users. Afterwards, the UI adaptation is tuned over several cycles in a process that includes end-user-evaluation, change-reporting and discussion, and readapting the UI based on the newly reported changes. He noted that the adaptation mechanism available to them in OFBiz supports reducing features (layout optimization is not supported) through XML configuration files, which are defined by the developers. Therefore, as he stated, the feedback mechanism provided by our approach is an important advantage, which empowers end-users to provide direct feedback to the system in order to shorten the cycles of the adaptation process. This reduces the implementation cost and allows the end-users to obtain an adapted UI more quickly. As a result of this interview, we were able to establish the process shown in Figure 7.7, which demonstrates conceptually these advantages.

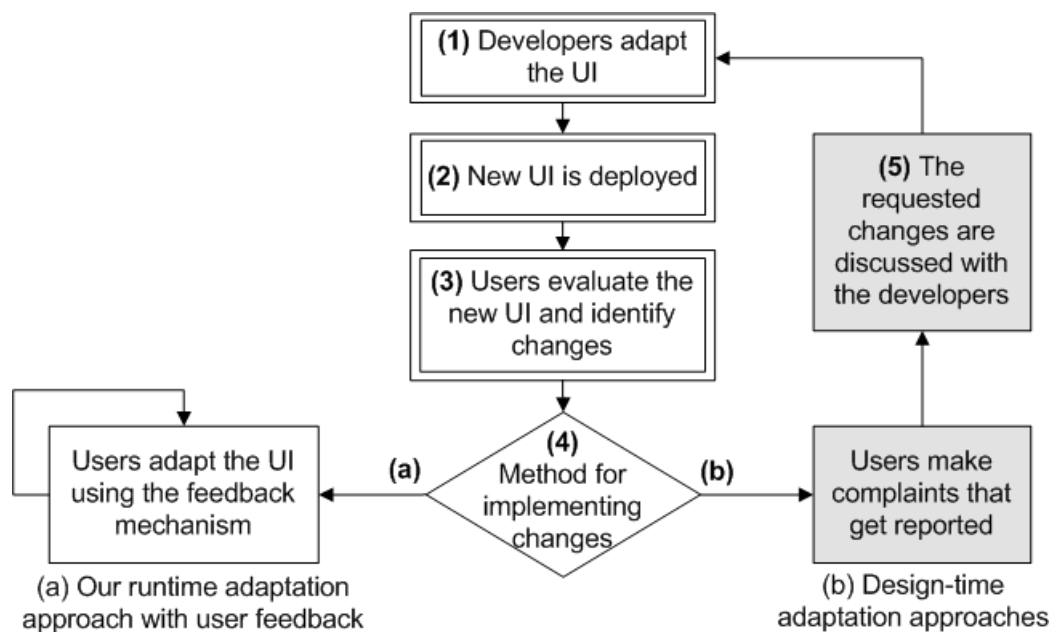


Figure 7.7: UI Adaptation Process: Design-Time versus Runtime UI Adaptation Cycles (based on interviewing an industry expert)

A complementary indication on the importance of runtime adaptation approaches is made by an existing research work, which states that software systems should attempt to break the boundary between development-time and runtime to handle the changes that cannot be anticipated or predicted beforehand (Baresi & Ghezzi 2010). Empowering end-users with control over

the UI adaptations narrows this boundary and helps in reducing the round trip in the adaptation process.

We estimated the time that the feedback mechanism could save in the UI adaptation cycle based on real-life data. We asked the manager who we interviewed to provide us with timestamps of requests on the different steps of the UI adaptation process from past projects. We were provided with a sample of 36 timestamps of requests from three past projects that were running in parallel. The timestamps were obtained by referring to historic emails of requests on development, deployment, and change reporting and discussion. Based on these timestamps, we calculated the mean number of days for developing and deploying the adapted UIs and reporting and discussing change requests between the enterprise employees and the software company. The results are shown in Figure 7.8 but the project names are hidden for confidentiality reasons. The results indicate that the highest mean days in the UI adaptation process are allocated to end-user evaluation, and change reporting and discussion (Project A=45.25, Project B=25.66, Project C=35) and a smaller mean number of days is allocated to the development and deployment of UI adaptations (Project A=9, Project B=4.75, Project C=5.25).

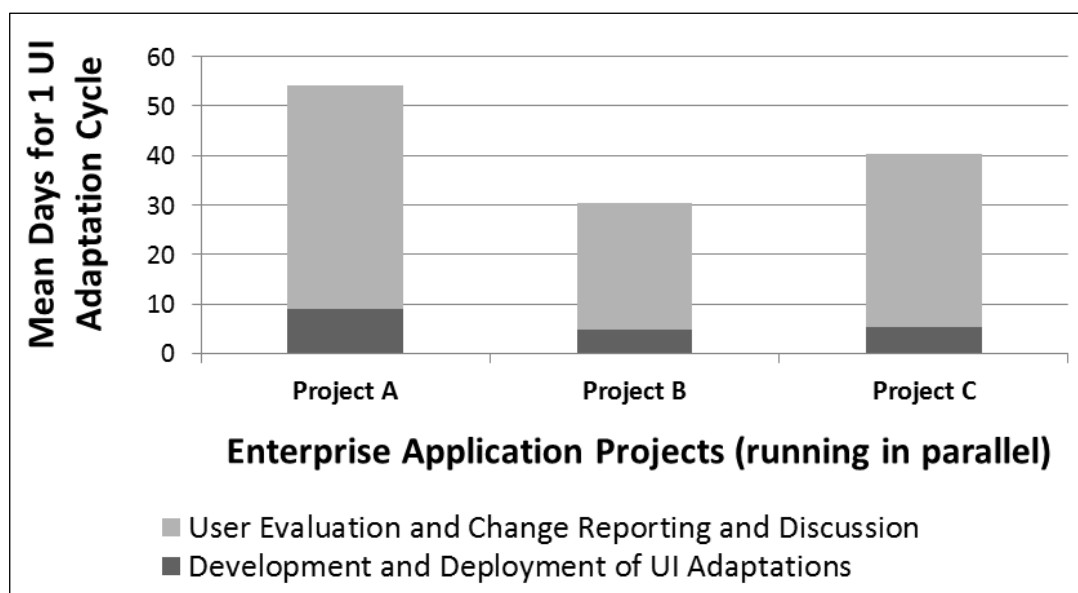


Figure 7.8: Mean Number of Days for One UI Adaptation Cycle from Three Real-Life Enterprise Application Projects Running in Parallel

The results shown in Figure 7.8 indicate that if the UI adaptation process was repeated from the start with every cycle, a period of over one month could pass before the end-users get their requested UI adaptations. On the other hand, if the end-users were given the ability to report the changes directly to the system through a feedback mechanism this process could become much shorter by eliminating the time required for development, deployment, and change discussion.

7.3.3 Discussing the Results of Interviewing the Industry Expert

This section provides a discussion of the results we got from interviewing an industry expert with respect to our UI adaptation approach. It also presents the threats to the validity of this part of the evaluation.

7.3.3.1 Discussion

The main intention of the interview was to inquire about the opinion of an industry expert on the generality and flexibility of our UI adaptation approach. However, the interviewee offered us interesting insights on the importance of the feedback loop in shortening the UI adaptation cycle by allowing end-users to report changes directly to the system rather than to the software company. Based on data from three real-life projects, we observed that when the end-users report UI adaptation changes to the software company, a period over one month could pass before they get the requested changes. Yet, with the feedback loop the changes can be obtained immediately.

We acknowledge the limitations of this interview due to the small amount of information that we were able to obtain on generality and flexibility. Therefore, we need to collect more data before being able to decide on whether to accept or reject the null hypothesis H_{0-3} , which was established in Chapter 3. The next section offers more details on the threats to validity.

7.3.3.2 Threats to Validity

The purpose of this interview is for providing an indication about the opinions of industry experts in our UI adaptation approach primarily on its generality and flexibility. We acknowledge however that interviewing more industry experts will support our claims further and can also offer us additional insights. Since the interview yielded more information on the feedback loop than any other characteristics of our approach, in the future we plan on running focus group sessions during which software developers can work with Cedar Studio and report their opinions on our approach's flexibility and learnability.

Concerning the UI adaptation cycle data illustrated in Figure 7.8, as we mentioned earlier it is based on a sample of 36 request timestamps from 3 projects. Therefore, our intention is not to generalize it but to give an indication about the time each adaptation cycle could take to show the usefulness of our runtime feedback mechanism in shortening these cycles.

7.4 Evaluation Based on Usability Studies

Prior research works such as Supple (Gajos et al. 2010), have shown that adaptive UIs can improve usability in certain scenarios. However, these works did not directly target enterprise application UIs, which can be more complex than others. Additionally, these works mostly focused on layout optimization and not feature-set minimization. Therefore, we conducted usability studies to determine whether RBUIS can significantly improve usability when used with real-life enterprise application UIs. One could expect an improvement especially when simplifying the feature-set of a UI, since the end-users are presented with a fewer number of fields. Yet, the main question behind our usability studies is on the significance of this improvement. Since implementing adaptive UIs can add some overhead in comparison to simply using one generic off-the-shelf UI, the significance of the improvement in usability is important. A marginal improvement might not be enough for enterprises to invest in adaptive UIs.

We conducted two usability studies, a preliminary online study and a more comprehensive one in the laboratory. We started with the online study because

it is less costly to run. Hence, in case it gave us negative indications we could perform adjustments on RBUIS before running a more thorough lab-based study. Our two usability studies aim to answer research question **Q8** by evaluating if the feature-set minimization and layout optimization provided by RBUIS can significantly improve the usability (efficiency, effectiveness, and satisfaction) of enterprise application UIs. Sections 7.4.1 and 7.4.2 describe the online study and lab-based one respectively, and present their outcomes.

7.4.1 Online Study

This study served as a preliminary low-cost evaluation of how much RBUIS can improve the usability of enterprise application UIs. The participants were presented with an interactive UI pair composed of an initial and an adapted version of the same interface. We selected the *Customer Maintenance* UI of the SAP ERP system. The initial version contains numerous tab-pages and a lot of fields. However, end-users with different roles in the enterprise require a different sub-set of the features. For example, some end-users require a simpler version for managing basic customer records. The simple version has a much smaller sub-set of the features available in the initial generic UI offered by SAP. The fields were selected based on existing information about the variability in SAP's user needs (Synactive GmbH 2010).

We developed a copy of SAP's UI alongside a simplified version containing the fields required for managing basic customer records. The initial and simplified UI versions are illustrated by Figure 7.9a, and Figure 7.9b respectively. The fields that are not required were hidden in the simplified UI version, and the widgets were regrouped accordingly. Some of the hidden fields were set to be reversible by the end-user, in order to test whether the participants are able to use the feedback mechanism for bringing back fields.

After completing the assigned task, the participants were asked to answer the System Usability Scale (SUS) questions shown in Section B.2 of Appendix B. The SUS questions allow us to measure the end-user satisfaction (perceived usability) and compare the results of the two UI versions. The time taken by

each participant to complete the task was also recorded to measure the efficiency and compare the results of the initial and the simplified UI versions.

(a) Initial Customer Maintenance User Interface

Address	Control Data	Payments	Marketing	Unloading Points	Export Data	Contacts
Name						
Title	Mr. <input type="text"/>					
Name	<input type="text"/>					
Search Terms						
Search Term 1/2	<input type="text"/>					
Street Address						
Street/House Number	<input type="text"/>					
Postal Code / City	<input type="text"/>					
Country	<input type="text"/>					
PO Box Address						
PO Box	<input type="text"/>					
Postal Code	<input type="text"/>					
Communication						
Language	<input type="text"/>					
Telephone	<input type="text"/>				Extension	<input type="text"/>
Fax	<input type="text"/>				Extension	<input type="text"/>
Data Line	<input type="text"/>					

(b) Simplified Customer Maintenance User Interface


Personal Information			
Title	Mr. <input type="text"/>		
Name	<input type="text"/>		
Search Term 1/2	<input type="text"/>		
Address			
Street/House Number	<input type="text"/>		
Postal Code / City	<input type="text"/>		
Country	<input type="text"/>		
Communication			
Telephone	<input type="text"/>	Extension	<input type="text"/>
Fax	<input type="text"/>	Extension	<input type="text"/>
Tax Information			
Tax Number	<input type="text"/>	Tax number type	<input type="text"/>
<input type="checkbox"/> Equalization Tax			
Bank Information			
Bank City	<input type="text"/>		
Bank Key	<input type="text"/>		
Bank Account	<input type="text"/>		
Payment Transactions			
Terms of payment	<input type="text"/>		
Tolerance group	<input type="text"/>		
Grouping Key	<input type="text"/>		
Export Credit Insurance			
Policy number	<input type="text"/>		
Institution number	<input type="text"/>		
Amount insured	<input type="text"/>		

Figure 7.9: Customer Maintenance UI Initial (a) and Simplified (b) Versions

7.4.1.1 Participant Recruitment and Demographics

We recruited 25 participants for this study using Amazon Mechanical Turk, a crowdsourcing internet marketplace. The existing literature has shown that Amazon Mechanical Turk is a viable option for conducting research studies (Paolacci et al. 2010). The site has the necessary elements for successfully completing a research project, and can provide participants with diverse demographic characteristics (Buhrmester et al. 2011). Each participant was paid a marginal amount of money (\$1) to participate in this study on which they spent an average time of 9 minutes and 53 seconds. To ensure the recruitment of serious participants, we requested Mechanical Turk workers who have completed more than 5000 hits (tasks) with over 95% accuracy.

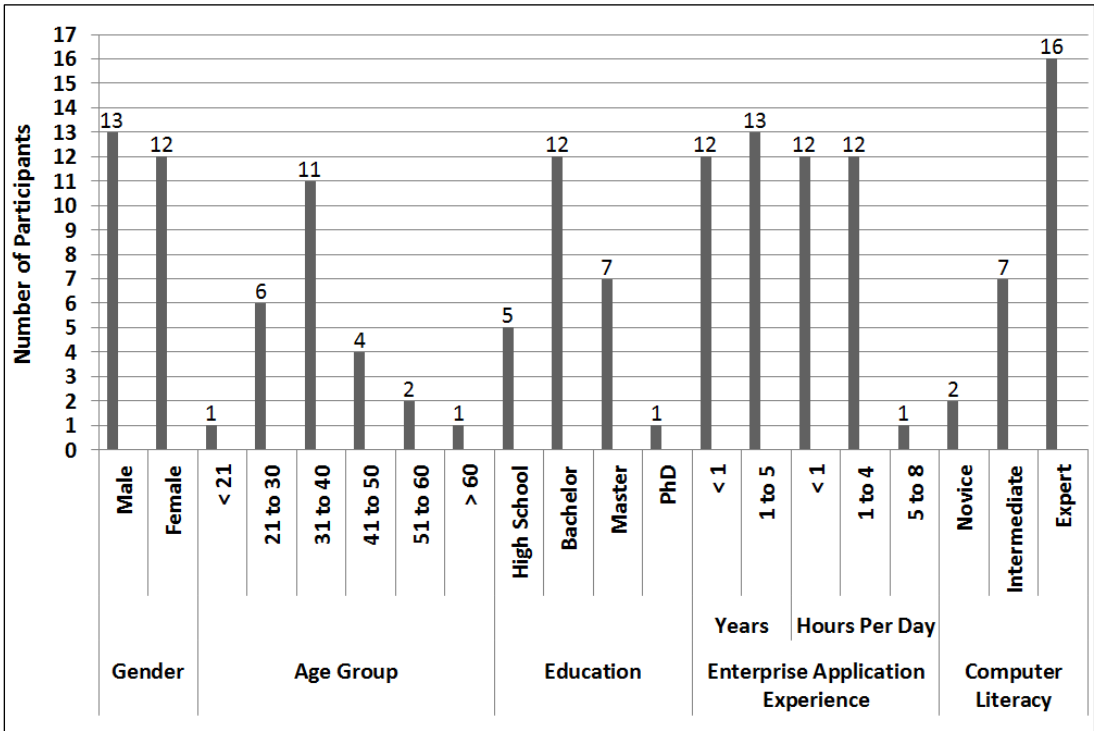


Figure 7.10: Participant Demographic Information for Online Usability Study

The participants were recruited at random and had diverse demographics as shown by the chart in Figure 7.10. None of them had previously used SAP’s *Customer Maintenance* UI. The demographic information was collected using the questionnaire presented in Section B.1 of Appendix B.

7.4.1.2 Task Allocated to Participants

The participants were asked to fill a set of fields required for creating a basic customer record using both UI versions. In the case of the simplified UI, two of the fields had to be retrieved through the user-feedback UI, thereby allowing us to test the participants' ability to use this feature. The instructions that were given to the participants are shown in Figure 7.11, and were displayed during the study on the right-hand side of the respective UI shown in Figure 7.9.

Instructions:

Consider that you need to input a customer record.


Please fill the following fields in the form on the left and answer the questions below.

Field	Value
Title	Mr.
Name	John Smith
Search Term 1/2	JoSm
Postal Code / City	MK4 545 / Milton Keynes
Country	United Kingdom
Bank city	London
Bank key	BK123987
Bank account	Current
Terms of payment	0001
Tolerance group	GRB1
Dunning procedure	DPROC1
Dunning level	4
Grouping key	GK1

Instructions:

Consider that you need to input a customer record.

Some fields are not visible on the screen.

You can click the green chameleon  icon in the top right corner to show the ones you need.

Please fill the following fields in the form on the left and answer the questions below.

Field	Value
Title	Mr.
Name	John Smith
Search Term 1/2	JoSm
Postal Code / City	MK4 545 / Milton Keynes
Country	United Kingdom
Bank city	London
Bank key	BK123987
Bank account	Current
Terms of payment	0001
Tolerance group	GRB1
Dunning procedure	DPROC1
Dunning level	4
Grouping key	GK1

Figure 7.11: Online Usability Study Participant Instructions for the Initial (Left) and Simplified (Right) User Interface Versions

In some cases, participants prefer the first UI option they see hence creating certain bias in a study's outcome. To avoid this potential bias, we presented half of the participants with the initial UI first and the other half with the simplified one first.

7.4.1.3 Results

We used a Wilcoxon signed-ranks test to test if there are statistically significant differences between the initial UI and the simplified one. This test is the nonparametric equivalent to the dependent t-test. The data that we collected did not have a normal distribution. Therefore, we used the Wilcoxon signed-rank test because it does not assume normality in the data. Also, this test can be applied to dependent variables that are continuous such as: the task completion times and SUS scores.

The results showed that the simplified UI elicited a statistically significant improvement in both perceived usability based on the SUS score ($Z = -3.530$, $p = 0.0004$) and efficiency based on the task completion time ($Z = -2.644$, $p = 0.008$). The asymptotic significance (2-tailed) is the p value for the test and the Wilcoxon signed-ranks test, is reported using the Z statistic. The p value is < 0.01 for both cases, indicating that the simplified UI version shows a very strong improvement over the initial one in terms of satisfaction and efficiency.

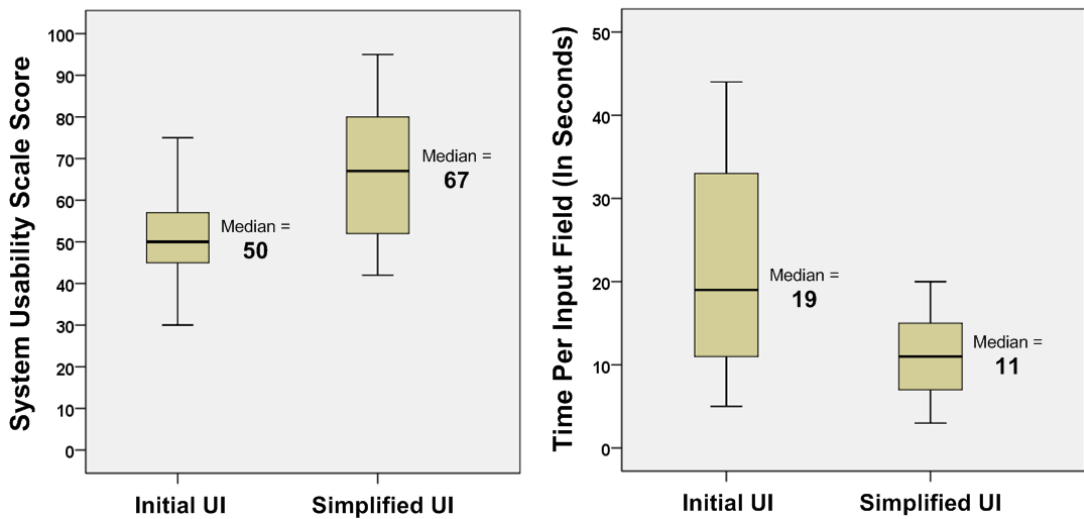


Figure 7.12: End-User Satisfaction and Efficiency Results

The results are illustrated by the box-plots in Figure 7.12. The means of the SUS score and time taken to complete the task are presented in Table 7.11 alongside the improvements. The improvements that the simplified UI offered were also reflected in the comments of some participants about it being more

efficient, whereas the initial UI made it too complicated to locate the fields that were dispersed across many tab-pages.

Table 7.6: Improvement in End-User Satisfaction and Efficiency after UI Simplification

	Mean SUS Score		
	Initial	Simplified	Improvement
	50.08 (se=4.005)	66.68 (se=3.496)	↑ 33.14 % (1.33×)
	Mean Task Completion Time (Per Input Field in Sec.)		
	Initial	Simplified	Improvement
	24.84 (se=3.794)	13.68 (se=2.369)	↓ 44.92 % (1.81×)

The ease-of-use of the feedback mechanism was reflected by the fact that 80% of the participants were able to use it by only referring to a few words of instruction on its purpose. Since 20% of the participants did not fill the two hidden fields, we assumed that they were not able to show them using the feedback mechanism.

Some participants also left one or two of the visible and required fields blank. Hence, the task completion time was calculated per input field rather than for the task as a whole. A limitation of conducting the study online is the inability to know the reason for leaving these fields blank. Hence, we only consider the results of this study as a preliminary indication of RBUIS's ability to improve usability. On the other hand, in the lab-based study we calculated the task completion time for the task as a whole. In case the participants left some fields blank, we can know if the reason was the complexity of the UI since the laboratory session is video-recorded. Hence, in the lab-based study we also calculated effectiveness in addition to efficiency and satisfaction.

7.4.2 Lab-Based Study

Since the online study (Section 7.4.1) showed an improvement in usability for simplified UIs, we conducted a more thorough lab-based study to check if it will yield similar results. This study has two parts, one on feature-set minimization and another on layout optimization.

In the feature-set minimization part, the participants were presented with two pairs of UIs on a desktop computer connected to a Tobii¹² eye-tracker. The UIs were selected from the SAP ERP system. The first pair included an initial and a simplified version of the Material UI shown in Figure 7.13a, and Figure 7.13b respectively. The second pair represented the Vendor maintenance UI, which is shown in its initial state in Figure 7.14a, and after simplification in Figure 7.14b.

Similar to the UI selected in the online study, the initial version of these UIs contains several tab-pages and a lot of fields, and can be simplified for roles that require more basic functionality. An existing document provides information and examples on the variation in SAP's user needs and helped us in determining what features to remove from the UIs (Synactive GmbH 2010). The fields that are not required by a role were hidden in the simplified UI version and the widgets were regrouped accordingly.

As shown by Figure 7.13a, and Figure 7.14a, each of the initial UIs has six tab-pages. As provided by SAP, these UIs originally have a larger number of tab-pages. However, we only used six because we wanted to keep the time required to complete the study reasonable for the participants. Nevertheless, a UI with six tab-pages offers enough complexity for evaluating the significance of usability improvement provided by feature-set minimization. Hence, if the simplified UI provided a significant improvement in comparison to an initial UI with six tab-pages, it is likely to yield similar or even better results when compared to an initial UI with more tab-pages.

In the part on layout optimization, the participants were also given two pairs of UIs. The first pair was presented to them on an iPad tablet and consisted of an initial and a simplified version of the Sales Transaction UI shown in Figure 7.15a, and Figure 7.15b respectively. This UI was selected from Microsoft's Dynamics ERP system. Its initial version has a desktop-style input grid and a lookup list for selecting items. Many enterprise systems maintain this UI style on tablets. The simplified UI provided a point-of-sale style sales transaction

¹² Tobii eye-tracker: www.tobii.com

with a buttons panel for selecting items and a grid with up/down arrows for changing an item's quantity. The second pair was presented on an HTC Desire mobile phone and consisted of an initial and a simplified version of the Contacts UI, which was previously shown in Chapter 5 by Figure 5.9.

Several measurements were made to compare the usability of both the initial and simplified versions of the UIs presented in this study. For both parts of the study, the participants were asked to answer the System Usability Scale (SUS) questions shown in Section B.2 of Appendix B to determine their satisfaction (perceived usability). In addition to SUS, the participants were asked to select three terms from the Microsoft Product Reaction Cards (Benedek & Miner 2002), shown in Section B.3 of Appendix B, to describe the UI. The time taken to complete each task was also recorded in both parts to measure the efficiency.

Additionally, in the feature-set minimization part of the study, eye-tracking was used to determine how lost the participants were in the initial UI versus the simplified one. The sessions were video-recorded, and the participants were asked to express their thoughts out loud in order to give us insights on their experience. The video recordings, especially in the feature-set minimization part, helped us in identifying whether a participant missed a field because of the UI's complexity or due to a simple human error.

(a) Initial Material User Interface

Basic	Sales Org.	General/Plant	Purchasing	Plant Data	Accounting
General Data					
Base Unit of Measure	ST	piece(s)	Material Group		
Old material number			Ext. Matl Group		
Division			Lab/Office		
Product allocation			Prod. hierarchy		
X-plant matl status			Valid form		
<input type="checkbox"/> Assign effect. vals			GenItemCatGroup		
Material authorization group					
Authorization Group					
Dimensions/EANs					
Gross Weight			Weight unit		
Net Weight					
Volume			Volume unit		
EAN/UPC			EAN Category		
Packaging material data					
Matl Grp Pack. Matls					
Ref. mat. for pkg					
Basic Data Texts					
Languages Maintained		0	Basic Data	Language	English

(b) Simplified Material User Interface

General Data			
Base Unit of Measure	ST	piece(s)	Material Group
Old material number			Lab/Office
Delivering Plant			Availability check
Tax ind. f. material			
Division			Currency
			EUR
Dimensions/EANs			
Gross Weight		Weight unit	
Net Weight			
EAN/UPC		EAN Category	
Packaging material data			
Matl Grp Pack. Matls			
Ref. mat. for pkg			
Sales / Purchasing			
LoadingGrp			GR Processing Time
Accounting			
Valuation Class			Price Unit
			Standard Price

Figure 7.13: Material UI Initial (a) and Simplified (b) Versions

(a) Initial Vendor User Interface

Initial	Address	Control	Purchasing Data	Accounting Info.	Payment
Name					
Title	Mr. <input type="text"/>				
Name	<input type="text"/>				
Search Terms					
Search term 1/2	<input type="text"/>				
Street Address					
Street/House number	<input type="text"/>				
Postal Code/City	<input type="text"/>				
Country	<input type="text"/>				
PO Box Address					
PO Box	<input type="text"/>				
Postal Code	<input type="text"/>				
Communication					
Language	English <input type="text"/> <input type="button" value="Other Comm..."/>				
Telephone	<input type="text"/>				
Fax	<input type="text"/>				
Data Line	<input type="text"/>				
Telebox	<input type="text"/>				

(b) Simplified Vendor User Interface

General Data	
Title	Mr. <input type="text"/>
Name	<input type="text"/>
Vendor	<input type="text"/>
Company Code	<input type="text"/>
Purchasing Organization	<input type="text"/>
Account group	<input type="text"/>
Search Terms	
Search term 1/2	<input type="text"/>
Street Address	
Street/House number	<input type="text"/>
Postal Code/City	<input type="text"/>
Country	<input type="text"/>
Accounting Information	
Recon. Account	<input type="text"/>
Payt Terms	<input type="text"/>
Purchasing data	
Planned deliv. time	<input type="text"/>
Confirmation Control	<input type="text"/>

Figure 7.14: Vendor UI Initial (a) and Simplified (b) Versions

(a) Initial Sales Transaction User Interface

Type/Type ID

Retail Inv.

Date

Document No.

Batch ID

Main

Customer ID

Default Site ID

Site1

Customer Name

Customer PO Number

Ship To Address

Currency ID

GBP

Add Item

ItemNo	Description	Qty	Price	Taxable	ExtendedPrice

Comment ID

Subtotal

Holds

Trade Discount

User-Defined

Freight

Distributions

Miscellaneous

Commissions

Tax

Total

(b) Simplified Sales Transaction User Interface

Drinks

Baked Goods

Customer Name

Select Customer

Shipping Address

Description	Qty	Price	Taxable	Ext. Price

Subtotal

Tax

Total

Figure 7.15: Sales Transaction UI Initial (a) and Simplified (b) Versions

7.4.2.1 Participant Recruitment and Demographics

We recruited 23 participants for this study by promoting it within The Open University community. The participants volunteered to take part without receiving any financial compensation. This study took an average of 45 minutes. The participants were recruited at random and had diverse demographics as shown by the chart in Figure 7.16. The demographic information was collected using the questionnaire presented in Section B.1 of Appendix B. None of the participants had previously used the UIs, which were selected for the study.

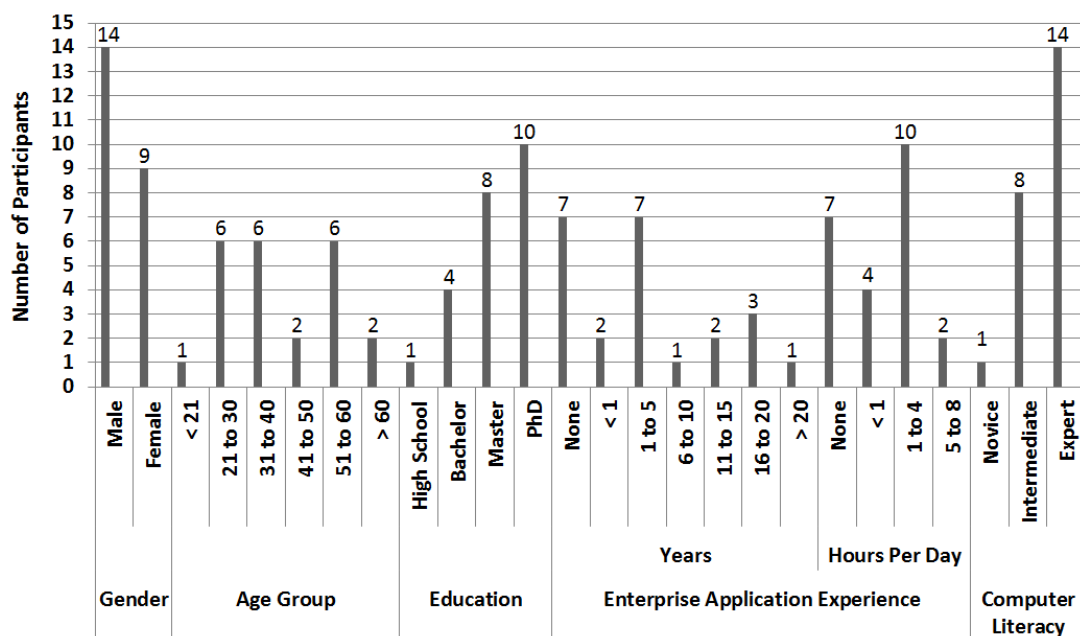


Figure 7.16: Participant Demographic Information for Lab-Based Usability Study

7.4.2.2 Tasks Allocated to Participants

In the feature-set minimization part, the participants were asked to fill a new record in each of the initial and simplified Material and Vendor UIs, following the instructions shown in Figure 7.17. The same instructions were given for both UI versions in each pair.

Instructions:

Consider that you need to input a new material record.

Please fill the following fields in the form on the left and then answer the questions below.

Field	Value
Old material number	123-456-789
Material Group	00207
Division	07
Gross Weight	16800
Weight Unit	KG
Net Weight	16800
Matl Grp Pack. Matls	M020
Delivering Plant	PM55
Tax ind. f. material	1
Availability check	02
GR Processing Time	30
LoadingGrp	0001
Valuation Class	3100
Price Unit	1
Standard Price	60

Instructions:

Consider that you need to input a new vendor record.

Please fill the following fields in the form on the left and then answer the questions below.

Field	Value
Title	Mr.
Name	John Smith
Search Term 1/2	JoSm
Vendor	v12345
Company Code	1000
Purchasing Organization	1000
Account Group	ZTMM
Planned delv. time	90
Confirmation Control	ANLI
Recon. account	16
Head office	16000
Payt Terms	0002

Figure 7.17: Lab-Based Usability Study Participant Instructions of the Feature-Set Minimization Part for the Material UI (Left) and Vendor UI (Right)

In the layout optimization part, the participants were asked to perform a simple task with each UI. For the first pair of UIs, they were asked to select a customer, add four items, and change the quantities for two of the added items as dictated by the following instructions:

- 1) Select the customer: Sophia Kenan
- 2) Add the following items:
 - Café Latte
 - Mocha
 - Blueberry Muffin
 - Apple Pie
- 3) Increase the quantity of the Café Latte item to 2
- 4) Increase the quantity of the Apple Pie item to 3

As for the second pair, the participants were asked to make a phone call to a contact called Charles Becker assuming that the UI is being used while running

down the street. They were presented with the ability to shake the phone, hence prompting it to change from one UI version to another.

Following the same practice as the online study, since some users have a tendency to like the first UI that they see, we presented half of the participants with the initial UI first and the other half with the adapted one first in order to avoid potential bias.

7.4.2.3 Satisfaction and Efficiency Results

We used a Wilcoxon signed-ranks test to check if there are statistically significant differences between the initial UIs and the simplified ones. Similar to the online usability study, the data that we collected did not have a normal distribution. Therefore, we used the Wilcoxon signed-rank test, nonparametric equivalent to the dependent t-test, because it does not assume normality in the data. Additionally, this test can be applied to continuous variables, which is our case with the task completion times and SUS scores.

The results presented in Table 7.7 show that simplifying the UI based on roles elicited a statistically significant improvement in both perceived usability based on the SUS score and efficiency based on the task completion time. The asymptotic significance (2-tailed) is the p value for the test and the Wilcoxon signed-ranks test, is reported using the Z statistic. The p value is < 0.01 for all cases, indicating that the simplified UI versions show a very strong improvement over the initial ones in terms of satisfaction and efficiency. We should note that the time taken by the participants to complete the task was not measured for the Contacts UI because the task is trivial and only takes a few seconds.

Table 7.7: Results of Wilcoxon Signed Ranks Test for Satisfaction and Efficiency

User Interface	Satisfaction (SUS Score)	Efficiency (Time)
Material	$Z = -4.200, p = 0.000027$	$Z = -4.197, p = 0.000027$
Vendor	$Z = -4.199, p = 0.000027$	$Z = -4.198, p = 0.000027$
Sales Transaction	$Z = -4.109, p = 0.000040$	$Z = -4.167, p = 0.000031$
Contacts	$Z = -2.617, p = 0.008877$	<i>Not Measured</i>

The results of the SUS scores are illustrated by the box-plots in Figure 7.18. We can observe three outliers but these are not extreme cases.

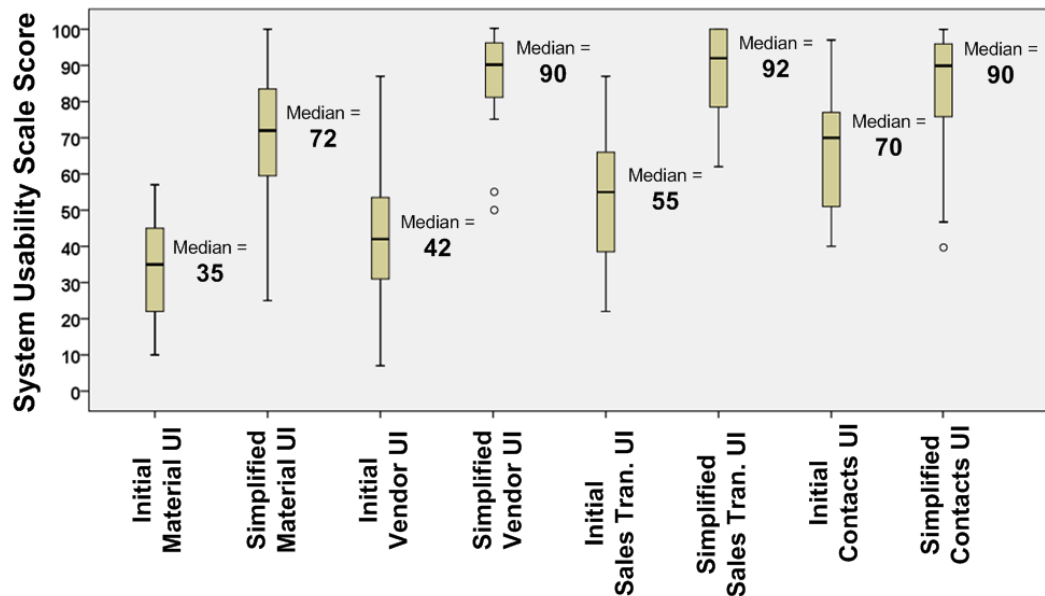


Figure 7.18: End-User Satisfaction Results for Lab-Based Usability Study

The mean SUS scores are presented in Table 7.8, alongside the improvement percentage for each of the four user interfaces used in the study. The improvement percentages show the advantage of the simplified UI versions over the initial ones for all cases.

Table 7.8: Improvement in End-User Satisfaction after UI Simplification

User Interface	Mean SUS Score		
	Initial	Simplified	Improvement
Material	34.09 (se=2.785)	68.78 (se=4.273)	↑ 101.76 % (2.01×)
Vendor	41.65 (se=3.942)	86.13 (se=2.788)	↑ 106.79 % (2.06×)
Sales Transaction	54.09 (se=4.488)	88.48 (se=2.830)	↑ 63.58 % (1.63×)
Contacts	66.70 (se=3.703)	83.74 (se=3.727)	↑ 25.55 % (1.25×)

The results of the time taken to complete the tasks are illustrated by the box-plots in Figure 7.19. We can observe three outliers but these are very close to the boundary of the box-plot.

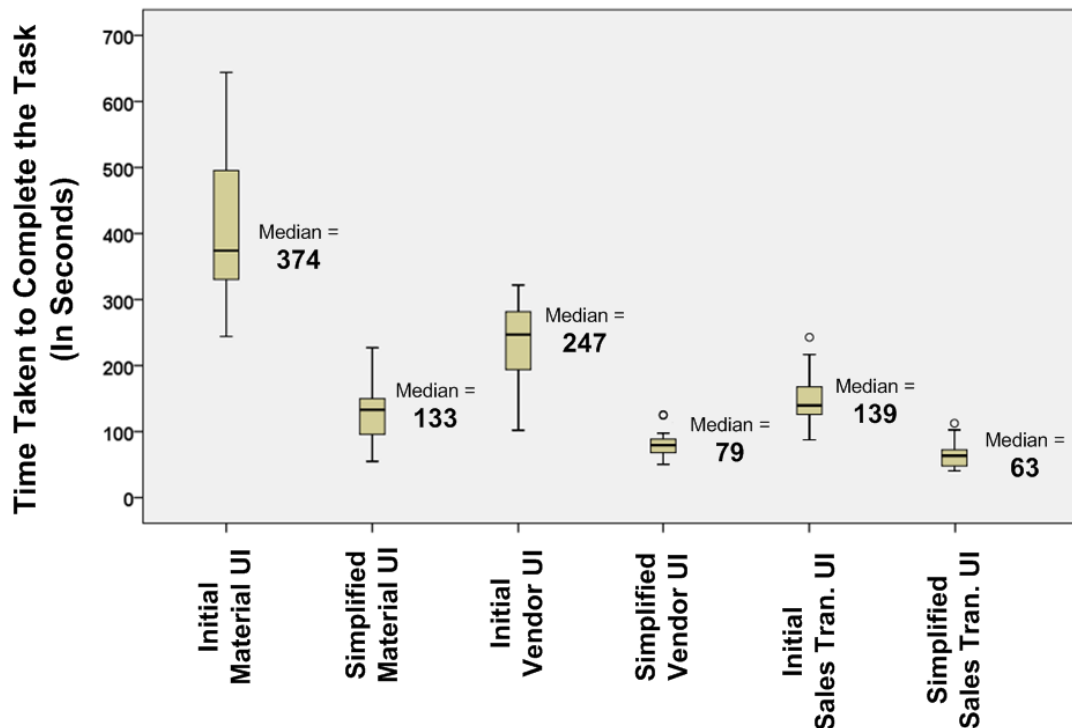


Figure 7.19: End-User Efficiency Results for Lab-Based Usability

The mean times taken to complete the allocated tasks are presented in Table 7.9, alongside the improvement percentage for each of the four user interfaces used in the study. The improvement percentages show the advantage of the simplified UI versions over the initial ones for all cases.

Table 7.9: Improvement in End-User Efficiency after UI Simplification

User Interface	Mean Task Completion Time (In Seconds)		
	Initial	Simplified	Improvement
Material	406.39 (se=23.005)	129.96 (se=9.010)	↓ 68.02 % (3.12×)
Vendor	236.30 (se=12.043)	84.57 (se=6.943)	↓ 64.21 % (2.79×)
Sales Transaction	148.87 (se=8.035)	63.83 (se=4.331)	↓ 57.12 % (2.33×)

The reaction cards selected by the participants to describe each UI confirm the improvement of end-user satisfaction achieved by the simplified UIs over the initial ones. The participants were asked to select three reaction cards to describe each UI from the list shown in Section B.3 of Appendix B. The pie charts illustrated in Figure 7.20 show that they selected a majority of positive terms when describing the simplified UIs, whereas they described the initial UIs with a majority of negative terms.

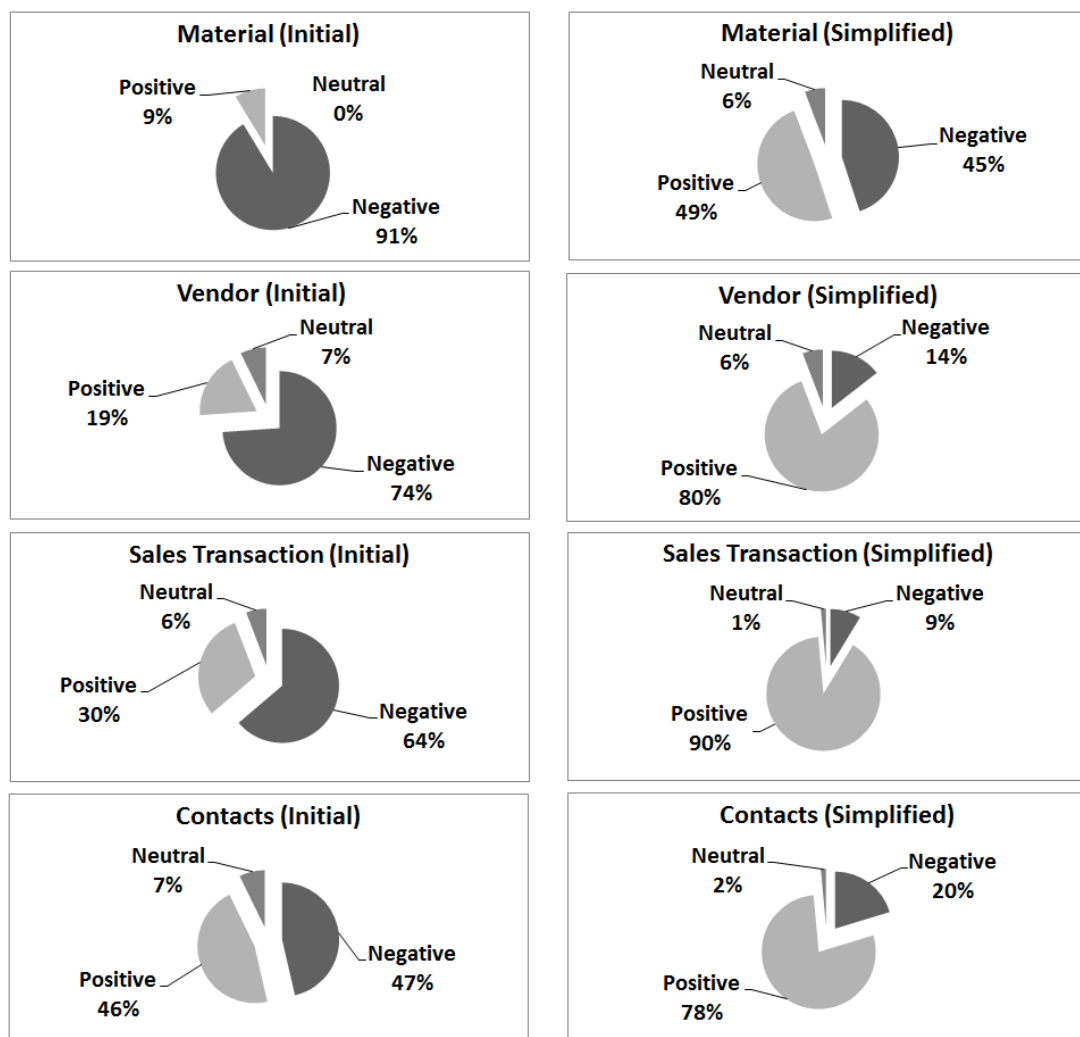


Figure 7.20: Aggregated Product Reaction Card Results for Lab-Based Usability Study

In addition to the reaction cards, the participants also mostly expressed dissatisfaction with the initial UIs in their verbal and written comments. For example, one participant said: *“I do not want a job where I have to use this UI*

(initial version), *whatever the job may be*". On the other, the simplified UIs mostly yielded positive comments. For example, although the task was exactly the same for both versions of the same UI one participant described a simplified UI as follows: *"The interface is simple, and the task is easier and more familiar"*.

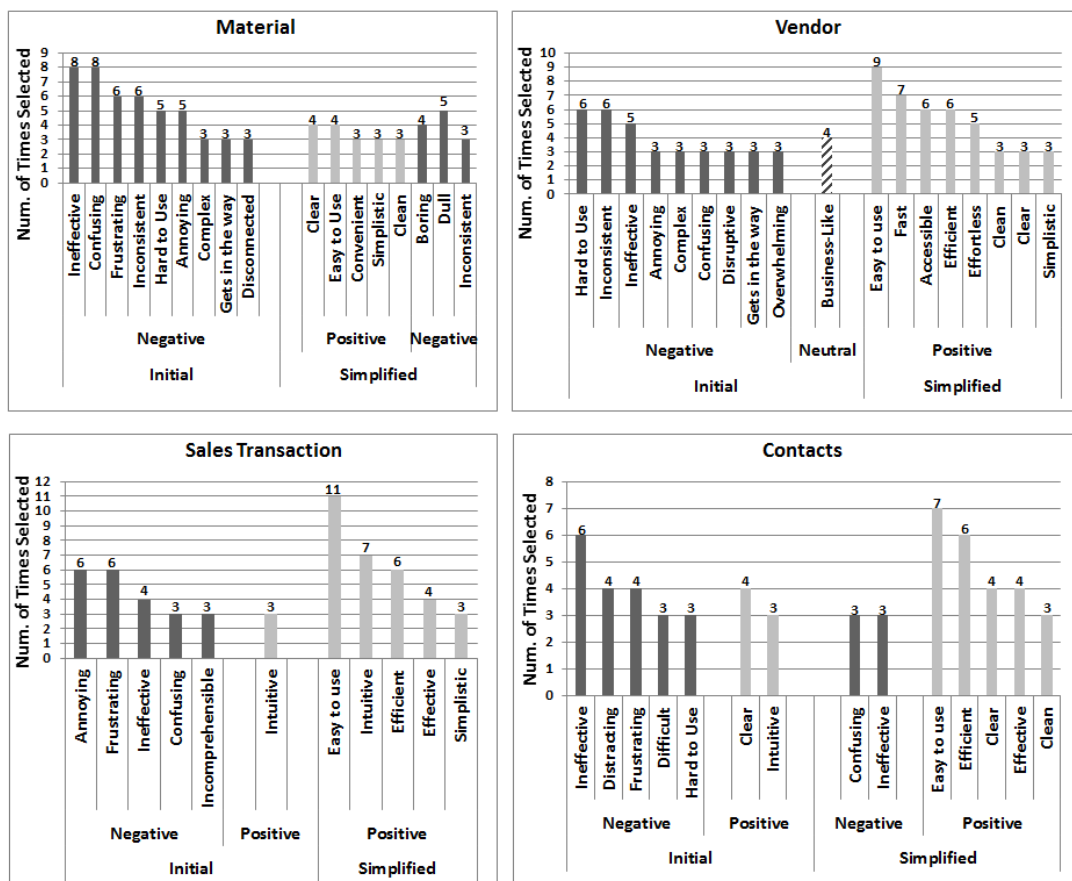


Figure 7.21: Product Reaction Cards Selected More than Two Times by Participants

The terms that were selected more than two times for each UI by the participants are illustrated by the bar-charts in Figure 7.21.

7.4.2.4 Effectiveness Results

The effectiveness is related to the *"accuracy and completeness with which specified users can achieve specified goals in particular environments"* (ISO 9241 2008). To measure and compare the effectiveness between the initial and simplified UIs, we checked the number of fields that were left blank by the participants. We were able to determine the reason behind the effectiveness

results because the sessions were video-recorded. This option was not available to us in the online study.

In the layout optimization part, the participants were able to complete the tasks successfully in most cases. The task given in the contacts UI (calling one of the contacts) was quite simple hence all the participants were able to perform it. In the Sales Transaction UI, very few participants missed entering one of the items or increasing a quantity. As pointed out by the participants themselves, this mistake was not due to the UIs but to a simple human error when reading the instructions.

The case was not the same for the feature-set minimization part where the participants left more blank fields with the initial UI than with the simplified one. A Wilcoxon signed-ranks test showed that the simplified UIs elicited a strong statistically significant improvement ($p < 0.01$) in both the Material ($Z = -2.728$, $p = 0.0063$) and the Vendor ($Z = -2.655$, $p = 0.0079$) UIs. The average percentages of the missing fields per participant are presented in Table 7.10.

Table 7.10: Improvement in End-User Effectiveness after UI Simplification

User Interface	Mean Missing Required Fields per Participant		
	Initial	Simplified	Improvement
Material	1.36 fields (6.33 %) (se=0.381)	0.14 fields (0.93 %) (se=0.100)	↓ 89.92 % (9.92×)
Vendor	0.95 fields (7.91 %) (se=0.259)	0.27 fields (2.25 %) (se=0.117)	↓ 71.57 % (3.51×)

Very few fields were missed with the simplified UIs constituting an average of 1.59% per participant. This percentage indicates that there is on average 1 missing field for every 5 participants. As we observed in the video-recordings, the main reason for missing a field in the simplified UI is a simple human error when reading the instructions (e.g., skipping a field by mistake).

The percentage of missing fields was higher with the initial UI versions with an average of 7.12%. This percentage indicates that there is on average 1 missing field for each participant. By observing the video-recordings we noticed that there were two main reasons behind the missing fields in the initial UI.

The first reason is that in some cases the participants tried to fill a field whenever they spotted it, thereby causing them to forget some fields because of not working sequentially. The second reason was knowingly skipping a field after getting frustrated from searching thoroughly and not finding it. Some participants tried using the search feature available in the web-browser to find certain fields. Yet, as we observed this was not helpful in all the cases since the participants still had to go through the tabs and apply the search on each one.

Another point to note about the initial UIs is that during the study we intervened a few times to offer the participants hints, primarily about two points: the existence of tab-pages, and the existence of scrolling in one of the tab-pages of the initial Vendor UI. Some participants did not notice that the initial UIs had tabs and some others did not notice that one of the tabs in the initial Vendor UI had a scroll bar. Hence, we gave hints 11 times on the tab-pages and 11 times on the scrolling. Some participants needed both hints; some needed only one, while the others did not require any of the two. On the other hand, in the simplified UIs no hints were needed for completing the task. Without the hints in the initial UIs, some participants might have taken longer to complete the task, and might have even left more required fields blank.

7.4.2.5 Eye-Tracking Results

The eye-tracking that we conducted in the feature-set minimization part of the study on the Material and Vendor UIs, helped us in determining and comparing how lost the participants were when using each of the UI versions. We discarded the eye-tracking data for 4 of our 23 participants because their eye-glasses prevented the eye-tracking device from recording any data. We used two metrics, namely the fixation duration and fixation count, from the eye-tracking data to determine how lost the participants were when using the different UI versions. The fixation duration is the measurement of how much time a participant spent focusing directly on certain points in the UI, while the fixation count is the measurement of the number of times that each participant directly focused on certain points.

The difference between the initial and simplified UIs in terms of fixation duration and fixation count is illustrated by the box-plots in Figure 7.22. By

observing the box-plots, we can immediately notice the improvement between the initial and simplified UI versions for all cases. These numbers represent the eye-tracking data of the part of the screen that shows the data entry form (SAP UI). The tasks shown in Figure 7.17 that were also displayed on the screen were excluded by defining an area-of-interest around each UI using the software tool provided by the Tobii eye-tracker. The areas-of-interest allow us to get accurate data about how much gazing each participant did on the UI itself.

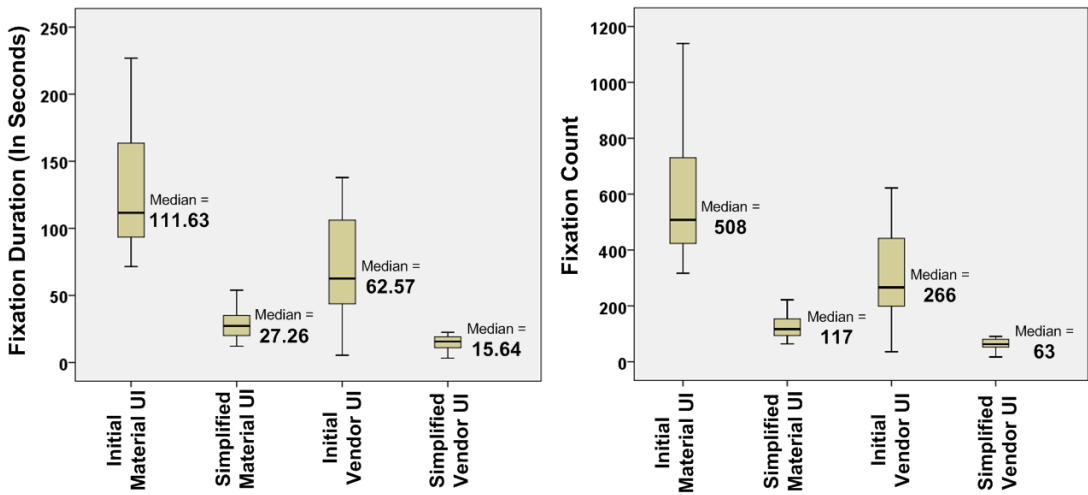


Figure 7.22: Eye-Tracking Results of Fixation Duration and Fixation Count

The mean values of the fixation duration and fixation count are presented in Table 7.11 alongside the improvement percentages, which show that the simplified UIs have a significant advantage over the initial ones.

Table 7.11: Improvement in Fixation Data after Simplification

User Interface	Mean Fixation Duration (In Seconds)		
	Initial	Simplified	Improvement
Material	137.68 (se=15.03)	29.77(se=2.748)	↓ 78.37 % (4.62×)
Vendor	71.98 (se=8.693)	14.97 (se=1.365)	↓ 79.20 % (4.80×)

User Interface	Mean Fixation Count		
	Initial	Simplified	Improvement
Material	599 (se=48.336)	128 (se=9.552)	↓ 78.63 % (4.67×)
Vendor	312 (se=34.755)	63 (se=5.217)	↓ 79.80 % (4.95×)

The heat maps in Figure 7.23 show the aggregated fixation data. We can visually observe the improvement provided by the simplified UI versions. We can notice that the highest amount of gazing is done on the left-hand-side of the input fields in the initial UI, where the labels are presented. This indicates that the participants were carefully checking the labels because it was difficult for them to find the fields in which they were required to enter data. On the other hand, with the simplified versions the overall gazing was much less intense.



Figure 7.23: Heat Maps Showing an Aggregation of the Participants' Gazing

The gaze plots in Figure 7.24, illustrate an example of one participant whose data came close to the means presented in Table 7.11. We can observe that even in an average case the significance of the improvement is visually noticeable.



Figure 7.24: Gaze Plots of One Participant with Data Close to the Mean

7.4.2.6 Additional Insights

Some participants expressed their preference of seeing the fields in the instructions in the same order as their counterparts in UI. We do acknowledge the importance of this point. However, we purposely presented the instruction fields in a different order than their UI counterparts to make the study closer to

a real-life scenario. For example, in real-life the data in the instructions can in some cases be received in an arbitrary order on paper from business partners such as: suppliers, customers, etc.

Some participants said that they disliked the fact that some of the labels were presented as acronyms. They said that having a full name for the field might be more helpful in searching for it. Some participants also stated that certain fields were not logically allocated under the tabs that they expected. For example, the General/Plant tab-page had the Shipping Times field in it. We should note that we copied the UIs as it is from SAP. Hence, these two points could also be improved upon in the initial design to make the UIs more usable.

7.4.3 Discussing the Results of the Usability Studies

This section discusses the results of the usability studies in terms of supporting or rejecting the null hypothesis H_{0-4} , which we established in Chapter 3, and indicates the limitations of the studies.

7.4.3.1 Discussion

Our initial online usability study yielded a very strong statistically significant improvement ($p < 0.01$) for the simplified UIs over the initial ones in terms of end-user satisfaction and efficiency. These results were encouraging hence we conducted a more thorough lab-based study to further investigate and validate this improvement.

The lab-based study also yielded a very strong statistically significant improvement ($p < 0.01$) for end-user satisfaction. As demonstrated earlier by the values in Table 7.8, feature-set minimization and layout optimization increased end-user satisfaction by an average of 104.27% and 44.56% respectively. These results are also confirmed by the reaction cards, which the participants selected to describe the UIs. As illustrated earlier in Figure 7.20, the participants mostly selected negative keywords to describe the initial UIs and positive ones to describe the simplified UIs. The implementation phase of enterprise applications could be very costly. This phase deals with the system deployment and includes: customization, data migration, training, etc. The end-

users' technology acceptance (satisfaction) is a key success factor in this phase (Section 1.3.3). Hence, offering the end-users UIs that they perceive as easy-to-use is vital for the success of the enterprise application. This point highlights the importance of the improvement in end-user satisfaction, which was yielded by the simplified UIs in the usability studies that we conducted.

A very strong statistically significant improvement ($p < 0.01$) was also demonstrated for end-user efficiency. Both feature-set minimization and layout optimization decreased the time taken to complete the tasks. As demonstrated earlier by the values in Table 7.9, feature-set minimization improved efficiency by an average of 66.11% and layout optimization improved it by an average of 57.12%. Enterprises generally pay large sums of money for their enterprise applications. Therefore, these applications are expected to have a good return on investment in terms of speeding up the daily business transactions. The significant improvement in efficiency achieved by the simplified UIs allows enterprises to have a quicker return on investment, instead of losing time and money with the less efficient initial user interfaces.

The improvement in end-user satisfaction and efficiency shown by the lab-based study, confirmed the results of the online study. Furthermore, additional work was done in the laboratory on measuring the end-users' effectiveness and also the extent to which they feel lost when trying to find the required fields in both versions of the user interface.

The lab-based study showed that the simplified UIs elicited a very strong statistically significant improvement ($p < 0.01$) in effectiveness over the initial ones. As shown earlier by the values in Table 7.10, the simplified UIs improved effectiveness by an average of 80.74%. Effectiveness is highly important for the activities managed by enterprise applications. Some errors committed by the end-users when entering financial information could cause the enterprise to lose a lot of time and money and might have legal repercussions due to discrepancies in some financial reports. Hence, the UIs should be simple enough to support the end-users to complete their tasks with as few errors as possible.

The eye-tracking results also demonstrated that feature-set minimization significantly improves the fixation duration and fixation count. We can notice

from the data that with the simplified UI versions the participants required a low fixation in order to find the fields that they needed for completing the assigned task. On the other hand, with the initial UI versions, the required fixation was much higher. As shown earlier by the values in Table 7.11, the average improvement is 78.78% and 79.21% for the fixation duration and fixation count respectively. Enterprise application end-users can spend several consecutive hours every day working with these systems. Hence, the more lost they feel when looking for UI elements, the more they have to fixate on certain points of the UI to try and locate them. This issue could degrade their performance even further after several hours of work, by having a negative effect on: efficiency due to getting tired, effectiveness due to being lost, and satisfaction due to frustration.

Based on the improvements in efficiency, effectiveness, and satisfaction, we can reject the null hypothesis $H_{0.4}$ and say that minimizing the feature-set and optimizing the layout of enterprise application UIs based on the context-of-use, significantly improves their usability in terms of: efficiency, effectiveness, and satisfaction. The results of the eye-tracking data also support our rejection of this null hypothesis even further.

7.4.3.2 Threats to Validity

One might ask about the effect of learning over time on the results, which we obtained from the usability studies. Would learning eventually make the end-users more effective, efficient, and satisfied with the initial UIs? We can say that as the end-users learn, their efficiency and effectiveness are likely to improve for both the initial and the simplified UIs. Learning however is unlikely to improve their satisfaction with the initial UIs. Subjecting the end-users to the complexity of the initial UIs could drive them to reject the enterprise application in the early stages of the training hence causing an implementation failure. An additional rationale for using the simplified UIs is that training the end-users on the initial UIs requires more time and money, whereas the simplified ones are likely to be learned more quickly.

7.5 Chapter Summary

In this chapter, we evaluated the contributions from both the technical and human perspectives. This evaluation helped us in answering the evaluation research questions that were defined in Section 3.1.2 of Chapter 3.

The first part of the evaluation, presented in Section 7.2, focused on the technical aspects of our contributions. We integrated RBUIS into an open-source enterprise application called OFBiz. Then, we established and applied several technical metrics to assess this integration. We were able to perform the integration by merely adding a few lines-of-code globally in OFBiz, thereby maintaining a low change-impact. We showed that there is a similarity between the UIs of OFBiz hence requiring fewer mapping rules for reverse-engineering them into a model-driven representation. This point makes the reverse-engineering process easier for enterprise applications, which contain thousands of user interfaces. Based on these results, we rejected the null hypothesis H_{0-1} because using the Cedar Architecture and interpreted runtime models allowed RBUIS to integrate in OFBiz without causing major changes to the way it functions or incurring a high integration cost. Furthermore, we conducted an efficiency and scalability test, which showed that our UI adaptation approach provides a real-time runtime performance and is scalable, thereby supporting our rejection of the null hypothesis H_{0-2} .

In the second part of the evaluation, presented in Section 7.3, we assessed our contributions based on industrial expertise and data from real-life projects. We conducted an interview with the manager of a software company, which builds products based on OFBiz in China, to inquire about his opinion on the generality and flexibility of our UI adaptation approach. Although, the interviewee acknowledged our approach's generality and flexibility, he was more interested in discussing its feedback mechanism. He offered us interesting insights on the importance of this mechanism for shortening the UI adaptation cycle by allowing end-users to report changes directly to the system rather than to the software company. We were able to estimate and demonstrate this improvement based on data from three real-life projects provided to us by the manager whom we interviewed. We acknowledge the limitations of this

interview due to the small amount of information that we were able to obtain on generality and flexibility. Therefore, we need to collect more data before being able to decide on whether to accept or reject the null hypothesis H_{0-3} .

The third and final part of the evaluation was presented in Section 7.4, and focused on whether our UI adaptation approach can significantly enhance the usability of enterprise applications UIs. We started this part of the evaluation by running a low-cost online usability study to compare an initial real-life enterprise application UI and a simplified version of it, in terms of end-user efficiency and satisfaction. This study showed that the simplified UI yielded a very strong statistically significant improvement ($p < 0.01$) over the initial one. Afterwards, we ran a more thorough lab-based usability study, which presented participants with four pairs of real-life enterprise UIs each containing an initial and a simplified version of the same UI. Two UI pairs were used for evaluating feature-set minimization, and two were used for evaluating layout optimization. The lab-based study confirmed the results on efficiency and satisfaction we obtained through the online study. Additionally, this study showed that the simplified UIs elicited a very strong statistically significant improvement ($p < 0.01$) in end-user effectiveness over the initial UIs. Furthermore, by applying eye-tracking in the feature-set minimization part of the study, we showed that participants were much more lost when using the initial UIs compared to using the simplified ones. Based on the results of the usability studies, we rejected the null hypothesis H_{0-4} since minimizing the feature-set and optimizing the layout of enterprise application UIs based on the context-of-use, significantly improved their usability in terms of: efficiency, effectiveness, and satisfaction.

8

Conclusions and Future Work

“The end of a melody is not its goal: but nonetheless, had the melody not reached its end it would not have reached its goal either. A parable.”

— Friedrich Nietzsche

This thesis presented a tool-supported approach for engineering adaptive model-driven enterprise application user interfaces. This chapter summarizes the work that was presented in the thesis and provides our concluding thoughts. It also presents a few points, which can be the target of future research endeavors. Since we already partially addressed some of these points, we summarize our preliminary results and offer some guidance for researchers wishing to tackle these challenges in the future.

8.1 Contributions

This thesis contributed an approach for engineering adaptive model-driven enterprise application user interfaces. In Chapter 2, we conducted a literature review of adaptive model-driven UI development systems. We evaluated the existing art after classifying it under: architectures, techniques, and tools. The evaluation was based on a set of criteria, which we compiled either based on direct recommendations from the literature or by combining features from multiple existing systems. The evaluation allowed us to identify the gaps, which offer potential for new research to be conducted. Based on these gaps, we defined our research questions and setup the corresponding hypotheses in Chapter 3. To answer the research questions, three technical contributions were presented and evaluated. These contributions are: the Cedar Architecture, the

Role-Based UI Simplification (RBUIS) mechanism, and their supporting IDE, Cedar Studio.

We presented the **Cedar Architecture** in Chapter 4, as a reference for the stakeholders interested in developing adaptive model-driven enterprise application UIs. This architecture is based on existing works including: the Three Layer Architecture and the CAMELEON Reference Framework. The Cedar Architecture has three server-side technology-independent layers. The *decision components* layer handles decision making in various adaptive UI scenarios such as evaluating whether a change in the context-of-use requires the UI to be adapted. The *adaptation components* layer is mainly responsible for adapting the UI models using the appropriate adaptive behavior. The *adaptive behavior and UI models* layer hosts the models that comprise the different levels of abstraction representing the UI, and the adaptive behavior that is applied to adapt the UI to the different contexts-of-use. The Cedar Architecture also has a *client components* layer. This layer hosts the technology-specific components, which are deployed to the client machine. These components are part of an API, which integrates in the enterprise application's code and allows it to connect to the server-side layers in order to adapt its user interfaces.

In Chapter 5, we presented **Role-Based User Interface Simplification** (RBUIS), a mechanism for improving usability through adaptive behavior by providing end-users with a minimal feature-set and an optimal layout based on the context-of-use. RBUIS merges role-based access control (RBAC) with adaptive behavior for simplifying UIs. In RBUIS, roles are divided into groups representing the aspects (e.g., computer literacy, job title, etc.) based on which the UI will be simplified. RBUIS supports *feature-set minimization* by assigning roles to task models for providing end-users with a minimal feature-set based on the context-of-use. The assignment could be done by IT personnel but there is also a potential for engaging end-users in the process. *Layout optimization* is supported by assigning roles to workflows that represent adaptive UI behavior visually and through code before being applied to CUI models. Furthermore, RBUIS promotes user-feedback for refining the adaptation operations. Hence, end-users are allowed to reverse feature-set minimizations and layout optimizations, and to choose possible alternative layout optimizations.

Cedar Studio was presented in Chapter 6 as an IDE for supporting the development of adaptive model-driven enterprise application UIs based on the Cedar Architecture and using RBUIS. Cedar Studio supports stakeholders such as: developers and IT personnel, in defining and managing artifacts such as: UI models and adaptive behavior, which are stored in a server-side database. This IDE can access the server-side layers of the Cedar Architecture through web-services in order to request or update artifacts. Cedar Studio supports visual-design tools for: (1) task models, (2) domain models, (3) AUI models, (4) CUI models (5) goal models, and (6) workflows. It also supports visual-design and code-editing tools for: (1) task-role assignments and RBUIS rules, (2) model constraints, and (3) dynamic scripts. Automatic generation and synchronization between the various levels of abstraction (task, AUI, and CUI models) is also supported, with the ability to make manual changes at any of these levels.

In Chapter 7, the contributions were evaluated from the technical and human perspectives. In the technical evaluation part, RBUIS was integrated into an existing open-source enterprise application called OFBiz. Several metrics were defined and applied to measure technical characteristics related to: reverse-engineering, integration, and runtime execution. We showed that RBUIS can be integrated into existing enterprise applications without causing major changes to the way they function or incurring a high integration cost. We also showed that it can run efficiently in real-time and that it is scalable. In the second part of the evaluation, we assessed our contributions based on industrial expertise and data from real-life projects. We obtained interesting insights on the importance of RBUIS's feedback mechanism in shortening the UI adaptation cycle, by allowing end-users to report their feedback directly to the system rather than to the software company. The third and final part focused on evaluating whether our UI adaptation approach can significantly improve the usability of enterprise application UIs. We conducted usability studies with real-life UIs. These studies showed that UIs with a minimized feature-set and an optimized layout elicited a very strong statistically significant improvement over their initial versions in terms of end-user efficiency, effectiveness, and satisfaction. Eye-tracking was also conducted, and it showed that minimizing the feature-set of complex UIs significantly decreases the extent to which end-users are lost when searching for input fields.

8.2 Future Work

In addition to the novel contributions that were made in this thesis, there is room for more work that could be the target of future endeavors. In this section, we present this potential future work and summarize the preliminary results of some of the points that we are proposing.

8.2.1 Preserving Designer Input on the User Interface

Designer input on the CUI model is important for the predictability and quality of the user interface. Hence, it would be better if designers can create a CUI model rather than completely generating it from an abstract model. Yet even though some approaches offer designers the ability to create CUIs, the designer's choices are bound to change upon adapting the UI according to the context-of-use. Nevertheless, in certain cases, there are some decisions that are hard to make automatically. For example, assume a UI element was hidden by applying a feature-set minimization operation. The layout refitting algorithm could have two choices for filling the gap left by the hidden element, either increasing the height of the element above it, or increasing the width of the element on its left-hand side. Such decisions could be better supported by the choices of the UI designers who know the functional nature of the interface.

We presented some preliminary work on a technique (Akiki et al. 2013c), which provides non-technical UI designers with a simple means of assigning constraints on the CUI. The constraints are taken into consideration and preserved when the UI is being automatically adapted to a particular context-of-use. Such constraints embody the characteristics of the UI that require human ingenuity and are not met by fully-automated techniques. More work is still required to make the proposed technique applicable in practice. A primary point would be devising an algorithm, which can convert explicit designer constraints into a constraint problem. This algorithm should then be utilized by the adaptation engine in combination with the algorithm (Listing 5.6) that refits the UI based on implicit constraints, in order to maintain the designer's input upon adapting the user interface.

8.2.2 Empowering New Design Participants

Leveraging the concept of crowdsourcing for adapting enterprise application UIs could be beneficial when considering the large communities and commercial interests behind these applications. In terms of RBUIS, crowdsourcing can allow end-users to adapt the feature-set using a simple tool without attaching the adaptations to user-roles. Afterwards, administrators could attach the UIs adapted by the crowd to one or more enterprise roles. This helps administrators in delegating some of the adaptation effort to the crowd.

We conducted initial research (Akiki et al. 2013b) in which we extended RBUIS by allowing end-users to perform feature-set minimization through a basic web-based feature-set editing tool. This tool can be made available online for an enterprise's community members. We evaluated the tool through a preliminary online user-study, which provided encouraging results in terms of end-user satisfaction, efficiency, and effectiveness. However, this study is limited in terms of the simplicity of the considered example. Additionally, it was carried out online hence we were not able to collect a lot of information on the participants' interaction with the tool. These limitations could be overcome in the future by conducting lab-based studies with more complex user interface examples. Furthermore, in the future, the web-based feature-set editing tool could be extended to support the adaptation of concrete UI widget properties (e.g., size, location, etc.). The study could also be enriched by testing the tool with a real-life enterprise application, and crowdsourcing the UI adaptations to that application's relevant online community.

8.2.3 Applying Simplification to Multiple Related User Interfaces

In enterprise applications, data input tasks can be scattered across multiple UIs. For example, consider the scenario of entering data for an inventory item in the Microsoft Dynamics Great Plains ERP system. The primary data entry is done using a UI called Item Maintenance. However, other item-related information is entered using separate UIs, which include: Internet Information, Options, Serial/Lot, Accounts, Currency, Print Options, Price List, Price Group, Purchasing Options, etc. On the domain model level, all these forms are linked

with the Item Maintenance UI through a primary-key/foreign-key relationship on the Item Reference field.

One could argue that all these forms could be developed at design-time, as one UI. Yet, we should consider the usage complexity that this design choice might incur in terms of the large number of UI widgets. Furthermore, we should consider the loading time of both the user interface elements and the data. Having a user interface that combines the required scattered fields in one UI, could enhance the performance of enterprise application end-users.

A possible solution could be extending RBUIS to support UI simplification in such scenarios, where related information is scattered across multiple UIs. A possible way of achieving this solution is by monitoring each end-user's behavior in terms of sequential usage of tasks. Afterwards, the system would suggest a new user interface, which combines the various related UI fragments that are being sequentially accessed by a particular end-user. This point can be achieved by suggesting a means for implementing the behavior monitors and behavior evaluators, which have been suggested by the Cedar Architecture.

8.3 Final Thoughts

Our approach is not intended to replace any of the stakeholders involved in the process of designing and developing user interfaces. It is merely meant to help them in producing user interfaces that fit the context-of-use better, thereby providing end-users with an improved usability.

As a final future outlook, we think that packaging adaptive model-driven user interface development systems as general-purpose products could increase their usefulness for real-life projects. This is already the case for existing commercial tools, which support the development of traditional UIs. Perhaps the initiative for producing an adaptive model-driven UI development system, which rivals traditional commercial IDEs, could be a joint venture between academics working in this area and industrial partners with a real interest in adopting this approach. Cedar Studio is a strong starting point for achieving this objective.

Bibliography

- (**Abrams et al. 1999**) Abrams, M. et al., 1999. UIML: An Appliance-Independent XML User Interface Language. *Computer Networks*, 31(11), pp.1695–1708. [http://dx.doi.org/10.1016/S1389-1286\(99\)00044-4](http://dx.doi.org/10.1016/S1389-1286(99)00044-4).
- (**Akiki 2010**) Akiki, P., 2010. Devising a New Model-Driven Framework for Developing GUI for Enterprise Applications. In G. A. Papadopoulos et al., eds. *Information Systems Development*. Boston, MA: Springer US, pp. 269–279. isbn:978-0-387-84809-9, 978-0-387-84810-5.
- (**Akiki et al. 2011**) Akiki, P.A. et al., 2011. *A Systematic Framework for Assessing the Implementation Phase of Enterprise Resource Planning Systems*, The Open University. <http://bit.ly/TR2012-06>.
- (**Akiki 2013**) Akiki, P.A., 2013. Engineering Adaptive User Interfaces for Enterprise Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. London, UK: ACM, pp. 151–154. <http://dx.doi.org/10.1145/2494603.2480333>.
- (**Akiki et al. 2015**) Akiki, P.A., Bandara, A.K. & Yu, Y., 2015. Adaptive Model-Driven User Interface Development Systems. *ACM Computing Surveys*, 47(1), pp.64:1–64:33 (In Press).
- (**Akiki et al. 2013a**) Akiki, P.A., Bandara, A.K. & Yu, Y., 2013. Cedar Studio: An IDE Supporting Adaptive Model-Driven User Interfaces for Enterprise Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. London, UK: ACM, pp. 139–144. <http://dx.doi.org/10.1145/2494603.2480332>.
- (**Akiki et al. 2013b**) Akiki, P.A., Bandara, A.K. & Yu, Y., 2013. Crowdsourcing User Interface Adaptations for Minimizing the Bloat in Enterprise Applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. London, UK: ACM, pp. 121–126. <http://dx.doi.org/10.1145/2494603.2480319>.
- (**Akiki et al. 2014**) Akiki, P.A., Bandara, A.K. & Yu, Y., 2014. Integrating Adaptive User Interface Capabilities in Enterprise Applications. In *Proceedings of the 36th International Conference on Software Engineering*. Hyderabad, India: IEEE/ACM, (forthcoming). <http://bit.ly/1fQ8aR7>.
- (**Akiki et al. 2013c**) Akiki, P.A., Bandara, A.K. & Yu, Y., 2013. Preserving Designer Input on Concrete User Interfaces Using Constraints While Maintaining Adaptive Behavior. In *Proceedings of the 2nd Workshop on*

- Context-Aware Adaptation of Service Front-Ends*. London, UK: CEUR-WS.org, pp. 9–16. <http://bit.ly/1mYQ2MA>.
- (Akiki et al. 2013d) Akiki, P.A., Bandara, A.K. & Yu, Y., 2013. RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. London, UK: ACM, pp. 3–12. <http://dx.doi.org/10.1145/2494603.2480297>.
- (Akiki et al. 2012) Akiki, P.A., Bandara, A.K. & Yu, Y., 2012. Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications. In *Proceedings of the 14th International Conference on Enterprise Information Systems*. Wroclaw, Poland: SciTePress, pp. 72–77. <http://dx.doi.org/10.5220/0003975800720077>.
- (Allbee 2008) Allbee, M., 2008. Lawson Smart Office brings WPF Goodness to the Enterprise. <http://bit.ly/1mdhYJ0> [Accessed August 1, 2013].
- (Appert & Beaudouin-Lafon 2006) Appert, C. & Beaudouin-Lafon, M., 2006. SwingStates: Adding State Machines to the Swing Toolkit. In *Proceedings of the 19th ACM Symposium on User Interface Software and Technology*. Montreux, Switzerland: ACM, pp. 319–322. <http://dx.doi.org/10.1002/spe.v38.11>.
- (Aquino et al. 2010) Aquino, N. et al., 2010. Usability Evaluation of Multi-Device/Platform User Interfaces Generated by Model-Driven Engineering. In *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement*. Bolzano-Bozen, Italy: ACM, pp. 30:1–30:10. <http://dx.doi.org/10.1145/1852786.1852826>.
- (Balme et al. 2004) Balme, L. et al., 2004. Cameleon-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In *Proceedings of the 2nd European Symposium on Ambient Intelligence*. Eindhoven, The Netherlands: Springer, pp. 291–302. http://dx.doi.org/10.1007/978-3-540-30473-9_28.
- (Balzert et al. 1996) Balzert, H. et al., 1996. The JANUS Application Development Environment-Generating More than the User Interface. In J. Vanderdonckt, ed. *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium: Presses Universitaires de Namur, pp. 183–206. isbn:2-87037-232-9.
- (Baresi & Ghezzi 2010) Baresi, L. & Ghezzi, C., 2010. The Disappearing Boundary Between Development-time and Run-time. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. Santa Fe, New Mexico, USA: ACM, pp. 17–22. <http://dx.doi.org/10.1145/1882362.1882367>.
- (Baresi et al. 2010) Baresi, L., Pasquale, L. & Spoletini, P., 2010. Fuzzy Goals for Requirements-Driven Adaptation. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*. Sydney, Australia: IEEE, pp. 125–134. <http://dx.doi.org/10.1109/RE.2010.25>.

- (**Bencomo et al. 2008**) Bencomo, N. et al., 2008. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *Proceedings of the 12th International Conference on Software Product Lines*. Limerick, Ireland: Lero Int. Science Centre, University of Limerick, pp. 23–32. <http://bit.ly/1iIEQRv>.
- (**Benedek & Miner 2002**) Benedek, J. & Miner, T., 2002. Measuring Desirability: New methods for Evaluating Desirability in a Usability Lab Setting. *Proceedings of Usability Professionals Association*, 2003, pp.8–12.
- (**Bergh et al. 2010**) Bergh, J., Sahni, D. & Coninx, K., 2010. Task Models for Safe Software Evolution and Adaptation. In D. England et al., eds. *Task Models and Diagrams for User Interface Design*. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 72–77. isbn:978-3-642-11796-1.
- (**Berti et al. 2004**) Berti, S. et al., 2004. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. In *Proceedings the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages*. Advanced Visual Interfaces. Gallipoli, Italy, pp. 103–110. <http://bit.ly/1hWhLFS>.
- (**Bihler & Mügge 2007**) Bihler, P. & Mügge, H., 2007. Supporting Cross-Application Contexts with Dynamic User Interface Fusion. In R. Koschke et al., eds. *Proceedings of INFORMATIK*. LNI. Bremen, Germany: GI, pp. 459–464. isbn:978-3-88579-203-1.
- (**Blouin et al. 2011**) Blouin, A. et al., 2011. Combining Aspect-Oriented Modeling with Property-Based Reasoning to Improve User Interface Adaptation. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Pisa, Italy: ACM, pp. 85–94. <http://dx.doi.org/10.1145/1996461.1996500>.
- (**Blouin & Beaudoux 2010**) Blouin, A. & Beaudoux, O., 2010. Improving Modularity and Usability of Interactive Systems with Malai. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York, USA: ACM, pp. 115–124. <http://dx.doi.org/10.1145/1822018.1822037>.
- (**Blumendorf 2009**) Blumendorf, M., 2009. *Multimodal Interaction in Smart Environments: A Model-based Runtime System for Ubiquitous User Interfaces*. PhD Thesis. Berlin, Germany: Technischen Universität Berlin. <http://bit.ly/1n2y0qs>.
- (**Blumendorf et al. 2006**) Blumendorf, M., Feuerstack, S. & Albayrak, S., 2006. Event-based Synchronization of Model-Based Multimodal User Interfaces. In A. Pleuss et al., eds. *Proceedings of the 2nd International Workshop on Model Driven Development of Advanced User Interfaces*. 9th International Conference on Model-Driven Engineering Languages and Systems. Genova, Italy: ACM, pp. 1–5. <http://bit.ly/1n2ysEY>.
- (**Blumendorf et al. 2008**) Blumendorf, M., Feuerstack, S. & Albayrak, S., 2008. Multimodal Smart Home User Interfaces. In K. Mukasa, A. Holzinger,

- & A. Karshmer, eds. *Proceedings of the Workshop on Intelligent User Interfaces for Ambient Assisted Living*. 13th International Conference on Intelligent User Interfaces. Gran Canaria, Spain: ACM. <http://bit.ly/1kEKkuD>.
- (Blumendorf et al. 2007)** Blumendorf, M., Feuerstack, S. & Albayrak, S., 2007. Multimodal User Interaction in Smart Environments: Delivering Distributed User Interfaces. In M. Mühlhäuser, A. Ferscha, & E. Aitenbichler, eds. *Constructing Ambient Intelligence*. Berlin: Springer-Verlag, pp. 113–120. isbn:978-3-540-85378-7, 978-3-540-85379-4.
- (Blumendorf et al. 2010)** Blumendorf, M., Lehmann, G. & Albayrak, S., 2010. Bridging Models and Systems at Runtime to Build Adaptive User Interfaces. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Berlin, Germany: ACM, pp. 9–18. <http://dx.doi.org/10.1145/1822018.1822022>.
- (Bodart et al. 1995)** Bodart, F. et al., 1995. Towards a Systematic Building of Software Architecture: The TRIDENT Methodological Guide. In P. A. Palanque & Rémi Bastide, eds. *Design, Specification and Verification of Interactive Systems*. Proceedings of the Eurographics Workshop. Toulouse, France: Springer, pp. 262–278. isbn:978-3-211-82739-0, 978-3-7091-9437-9.
- (Botterweck 2011)** Botterweck, G., 2011. Multi Front-End Engineering. In H. Hussmann, G. Meixner, & D. Zuehlke, eds. *Model-Driven Development of Advanced User Interfaces*. Springer, pp. 27–42. isbn:978-3-642-14561-2, 978-3-642-14562-9.
- (Brdiczka et al. 2007)** Brdiczka, O., Crowley, J.L. & Reignier, P., 2007. Learning Situation Models for Providing Context-Aware Services. In C. Stephanidis, ed. *Universal Access in HCI*. Springer-Verlag, pp. 23–32. isbn:978-3-540-73280-8, 978-3-540-73281-5.
- (Brooke 1996)** Brooke, J., 1996. SUS: A Quick and Dirty Usability Scale. In P. W. Jordan et al., eds. *Usability Evaluation in Industry*. London, UK: Taylor and Francis. <http://bit.ly/1jicBsB>.
- (Buhrmester et al. 2011)** Buhrmester, M., Kwang, T. & Gosling, S.D., 2011. Amazon’s Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data? *Perspectives on Psychological Science*, 6(1), pp.3–5. <http://dx.doi.org/10.1177/1745691610393980>.
- (Calvary et al. 2003)** Calvary, G. et al., 2003. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15(3), pp.289–308. [http://dx.doi.org/10.1016/S0953-5438\(03\)00010-9](http://dx.doi.org/10.1016/S0953-5438(03)00010-9).
- (Calvary et al. 2005)** Calvary, G. et al., 2005. Towards a New Generation of Widgets for Supporting Software Plasticity: The ”Comet”. In R. Bastide, P. Palanque, & J. Roth, eds. *Engineering Human Computer Interaction and Interactive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 306–324. isbn:978-3-540-26097-4, 978-3-540-31961-0.

- (**Cao 2008**) Cao, Y., 2008. Pareto Front - MATLAB Central. <http://bit.ly/ParetoFrontAlg> [Accessed December 19, 2012].
- (**Carroll & Carrithers 1984**) Carroll, J.M. & Carrithers, C., 1984. Training Wheels in a User Interface. *Communications of the ACM*, 27(8), pp.800–806. <http://dx.doi.org/10.1145/358198.358218>.
- (**Cheng et al. 2009**) Cheng, B.H.C. et al., 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In B. H. C. Cheng et al., eds. *Software Engineering for Self-Adaptive Systems*. Springer, pp. 1–26. isbn:978-3-642-02160-2, 978-3-642-02161-9.
- (**Clerckx et al. 2006**) Clerckx, T. et al., 2006. A Task-Driven User Interface Architecture for Ambient Intelligent Environments. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*. Sydney, Australia: ACM, pp. 309–311. <http://dx.doi.org/10.1145/1111449.1111520>.
- (**Clerckx et al. 2005**) Clerckx, T., Luyten, K. & Coninx, K., 2005. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In R. Bastide, P. A. Palanque, & J. Roth, eds. *Engineering Human Computer Interaction and Interactive Systems*. Springer, pp. 77–95. isbn:978-3-540-26097-4, 978-3-540-31961-0.
- (**Cohen 1988**) Cohen, J., 1988. *Statistical Power Analysis for the Behavioral Sciences* 2nd ed., Routledge Academic. isbn:0805802835.
- (**Coninx et al. 2003**) Coninx, K. et al., 2003. Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In *Proceedings of the 5th International Symposium on Human-Computer Interaction with Mobile Devices and Services*. Udine, Italy: Springer, pp. 256–270. isbn:978-3-540-40821-5, 978-3-540-45233-1.
- (**Coutaz 2010**) Coutaz, J., 2010. User Interface Plasticity: Model Driven Engineering to the Limit! In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Berlin, Germany: ACM, pp. 1–8. <http://dx.doi.org/10.1145/1822018.1822019>.
- (**Coyette & Vanderdonckt 2005**) Coyette, A. & Vanderdonckt, J., 2005. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In *Proceedings of 10th IFIP TC 13 International Conference on Human-Computer Interaction*. Rome, Italy: Springer, pp. 12–16. isbn:978-3-540-28943-2, 978-3-540-31722-7.
- (**Crease et al. 2000**) Crease, M., Brewster, S. & Gray, P., 2000. Caring, Sharing Widgets: A Toolkit of Sensitive Widgets. In *Proceedings of the 14th British Computer Society Human Computer Interaction Conference*. London, United Kingdom: Springer, pp. 257–270. http://dx.doi.org/10.1007/978-1-4471-0515-2_17.
- (**Demeure et al. 2009**) Demeure, A. et al., 2009. Design by Example of Graphical User Interfaces Adapting to Available Screen Size. In V. Lopez-Jaquero et al., eds. *Proceedings of the 7th International Conference on*

- Computer-Aided Design of User Interfaces*. Albacete, Spain: Springer-Verlag, pp. 277–282. isbn:978-84882-205-4.
- (Demeure et al. 2008)** Demeure, A., Calvary, G. & Coninx, K., 2008. COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. In T. C. Graham & P. Philippe, eds. *Proceedings of the 15th International Workshop on Interactive Systems Design Specification and Verification*. Kingston, Canada: Springer, pp. 225 – 237. http://dx.doi.org/10.1007/978-3-540-70569-7_21.
- (Dragicevic & Fekete 2001)** Dragicevic, P. & Fekete, J.-D., 2001. Input Device Selection and Interaction Configuration with ICON. In A. Blandford, J. Vanderdonckt, & P. Gray, eds. *People and Computers XV—Interaction without Frontiers*. Springer, pp. 543–558. <http://bit.ly/1iAzpVD>.
- (Duarte & Carriço 2006)** Duarte, C. & Carriço, L., 2006. A Conceptual Framework for Developing Adaptive Multimodal Applications. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*. Sydney, Australia: ACM, pp. 132–139. <http://dx.doi.org/10.1145/1111449.1111481>.
- (Easterbrook et al. 2008)** Easterbrook, S. et al., 2008. Selecting Empirical Methods for Software Engineering Research. In F. Shull, J. Singer, & D. Sjöberg, eds. *Guide to Advanced Empirical Software Engineering*. London: Springer London, pp. 285–311. isbn:978-1-84800-043-8.
- (Elwert & Schlunbaum 1995)** Elwert, T. & Schlunbaum, E., 1995. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In P. A. Palanque & R. Bastide, eds. *Proceedings of the Eurographics Workshop on Design, Specification and Verification of Interactive Systems*. Toulouse, France: Springer, pp. 193–208. isbn:978-3-211-82739-0, 978-3-7091-9437-9.
- (Ferraiolo et al. 2001)** Ferraiolo, D.F. et al., 2001. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3), pp.224–274. <http://dx.doi.org/10.1145/501978.501980>.
- (Feuerstack 2008)** Feuerstack, S., 2008. *A Method for the User-centered and Model-based Development of Interactive Application*. PhD Thesis. Berlin, Germany: Technischen Universität Berlin. <http://bit.ly/1krVe6C>.
- (Feuerstack et al. 2008)** Feuerstack, S. et al., 2008. Model-based Layout Generation. In P. Bottoni & S. Levialdi, eds. *Proceedings of the Working Conference on Advanced Visual Interfaces*. Napoli, Italy: ACM, pp. 217–224. <http://dx.doi.org/10.1145/1385569.1385605>.
- (Feuerstack, Blumendorf, Lehmann, et al. 2006)** Feuerstack, S. et al., 2006. Seamless Home Services. In A. Maña & V. Lotz, eds. *Developing Ambient Intelligence*. Paris, France: Springer, pp. 1–10. isbn:978-2-287-47469-9, 978-2-287-47610-5.

- (**Feuerstack, Blumendorf & Albayrak 2006**) Feuerstack, S., Blumendorf, M. & Albayrak, S., 2006. Bridging the Gap between Model and Design of User Interfaces. In R. L. Christian Hochberger, ed. *GI Jahrestagung (2)*. GI-Edition - Lecture Notes in Informatics. Dresden, Germany: GI, pp. 131–137. <http://bit.ly/R6nUto>.
- (**Findlater & McGrenere 2007**) Findlater, L. & McGrenere, J., 2007. Evaluating Reduced-Functionality Interfaces According to Feature Findability and Awareness. In *Proceedings of the 13th International Conference on Human-Computer Interaction*. Springer, pp. 592–605. isbn:3-540-74794-X, 978-3-540-74794-9.
- (**Florins 2006**) Florins, M., 2006. *Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces*. PhD Thesis. Louvain, Belgium: Université Catholique de Louvain. <http://bit.ly/1fvVeW5>.
- (**Florins & Vanderdonckt 2004**) Florins, M. & Vanderdonckt, J., 2004. Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. Funchal, Madeira, Portugal: ACM, pp. 140–147. <http://dx.doi.org/10.1145/964442.964469>.
- (**Foley et al. 1991**) Foley, J. et al., 1991. GUIDE – An Intelligent User Interface Design Environment. In J. Sullivan & S. Tyler, eds. *Architectures for Intelligent Interfaces: Elements and Prototypes*. Addison-Wesley, pp. 339–384. isbn:0-201-50305-0.
- (**Fonseca 2010**) Fonseca, J.M.C., 2010. Model-Based UI XG Final Report. <http://bit.ly/ModelBasedUIXGFinalReport> [Accessed May 30, 2012].
- (**France & Rumpe 2007**) France, R. & Rumpe, B., 2007. Model-Driven Development of Complex Software: A Research Roadmap. In *Proceedings of the Workshop on the Future of Software Engineering*. International Conference on Software Engineering. Minneapolis, USA: IEEE, pp. 37–54. <http://dx.doi.org/10.1109/FOSE.2007.14>.
- (**Frankel 2003**) Frankel, D., 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley. isbn:9780471319207.
- (**Gajos & Weld 2005**) Gajos, K. & Weld, D.S., 2005. Preference Elicitation for Interface Optimization. In *Proceedings of the 18th ACM Symposium on User Interface Software and Technology*. Seattle, USA: ACM, pp. 173–182. <http://dx.doi.org/10.1145/1095034.1095063>.
- (**Gajos et al. 2010**) Gajos, K.Z., Weld, D.S. & Wobbrock, J.O., 2010. Automatically Generating Personalized User Interfaces with Supple. *Artificial Intelligence*, 174(12), pp.910–950. <http://dx.doi.org/10.1016/j.artint.2010.05.005>.
- (**Gajos et al. 2007**) Gajos, K.Z., Wobbrock, J.O. & Weld, D.S., 2007. Automatically Generating User Interfaces Adapted to Users' Motor and Vision Capabilities. In *Proceedings of the 20th ACM Symposium on User*

- Interface Software and Technology*. Newport, Rhode Island, USA: ACM, pp. 231–240. <http://dx.doi.org/10.1145/1294211.1294253>.
- (**Galvao & Goknil 2007**) Galvao, I. & Goknil, A., 2007. Survey of Traceability Approaches in Model-Driven Engineering. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*. Annapolis, Maryland, USA: IEEE, pp. 313–326. isbn:0-7695-2891-0.
- (**García Frey et al. 2012**) García Frey, A. et al., 2012. UsiComp: An Extensible Model-Driven Composer. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark: ACM, pp. 263–268. <http://dx.doi.org/10.1145/2305484.2305528>.
- (**García Frey et al. 2010**) García Frey, A., Calvary, G. & Dupuy-Chessa, S., 2010. Xplain: An Editor for Building Self-Explanatory User Interfaces by Model-Driven Engineering. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Berlin, Germany: ACM, pp. 41–46. <http://dx.doi.org/10.1145/1822018.1822026>.
- (**Garlan et al. 2004**) Garlan, D. et al., 2004. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10), pp.46–54. <http://dx.doi.org/10.1109/MC.2004.175>.
- (**Gartner 2013**) Gartner, 2013. *Gartner's Forecast Analysis: Enterprise Application Software, Worldwide, 2011-2016, 4Q12*, <http://gtmr.it/1fSE9Am> [Accessed April 1, 2014].
- (**Green 1985**) Green, M., 1985. The University of Alberta User Interface Management System. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. San Francisco, California: ACM, pp. 205–213. <http://dx.doi.org/10.1145/325334.325286>.
- (**Griffiths et al. 2001**) Griffiths, T. et al., 2001. Teallach: A Model-Based User Interface Development Environment for Object Databases. *Interacting with Computers*, 14(1), pp.31–68. [http://dx.doi.org/10.1016/S0953-5438\(01\)00042-X](http://dx.doi.org/10.1016/S0953-5438(01)00042-X).
- (**Guerrero-Garcia et al. 2009**) Guerrero-Garcia, J. et al., 2009. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proceedings of the 2009 Latin American Web Congress*. Merida, Yucatan, Mexico: IEEE, pp. 36–43. <http://dx.doi.org/10.1109/LA-WEB.2009.40>.
- (**Hayes et al. 1985**) Hayes, P.J., Szekely, P.A. & Lerner, R.A., 1985. Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In *Proceedings of the 3rd SIGCHI Conference on Human Factors in Computing Systems*. San Francisco, California: ACM, pp. 169–175. <http://dx.doi.org/10.1145/317456.317488>.
- (**Hill 1986**) Hill, R.D., 1986. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction—the Sassafras UIMS. *ACM Transactions on Graphics*, 5(3), pp.179–210. <http://dx.doi.org/10.1145/24054.24055>.

- (**Huebscher & McCann 2008**) Huebscher, M.C. & McCann, J.A., 2008. A Survey of Autonomic Computing—Degrees, Models, and Applications. *ACM Computing Surveys*, 40(3), pp.7:1–7:28. <http://dx.doi.org/10.1145/1380584.1380585>.
- (**Huxham et al. 1986**) Huxham, F.A., Burnard, D. & Takatsuka, J., 1986. *Using the Macintosh Toolbox with C*, SYBEX. isbn:978-0895885722.
- (**IBM 2006**) IBM, 2006. An Architectural Blueprint for Autonomic Computing. <http://bit.ly/MapeKLoop> [Accessed April 5, 2013].
- (**ISO 9241 2008**) ISO 9241, 2008. ISO 9241-12:1998 - Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) -- Part 12: Presentation of information. <http://bit.ly/ISO9214> [Accessed May 14, 2012].
- (**Jabarin & Graham 2003**) Jabarin, B. & Graham, T.C.N., 2003. Architectures for Widget-Level Plasticity. In J. Jorge, N. Jardim Nunes, & J. Falcão e Cunha, eds. *Interactive Systems. Design, Specification, and Verification*. Springer, pp. 124–138. isbn:978-3-540-20159-5.
- (**Jacob 1986**) Jacob, R.J., 1986. A Specification Language for Direct-Manipulation User Interfaces. *ACM Transactions on Graphics*, 5(4), pp.283–317. <http://dx.doi.org/10.1145/27623.27624>.
- (**Jacobson et al. 2007**) Jacobson, S. et al., 2007. *The ERP Market Sizing Report, 2006–2011*, Boston, MA: AMR Research, Inc. <http://bit.ly/1rS02XA> [Accessed April 2, 2012].
- (**Janssen et al. 1993**) Janssen, C., Weisbecker, A. & Ziegler, J., 1993. Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands: ACM, pp. 418–423. <http://dx.doi.org/10.1145/169059.169335>.
- (**Johnson 1992**) Johnson, J., 1992. Selectors: Going Beyond User-Interface Widgets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Monterey, California, USA: ACM, pp. 273–279. <http://dx.doi.org/10.1145/142750.142810>.
- (**Judith Hurwitz 2007**) Judith Hurwitz, 2007. How does SAP turn 250 million lines of code into modular services? <http://bit.ly/SAPLinesOfCode> [Accessed August 1, 2013].
- (**Kawai et al. 1996**) Kawai, S., Aida, H. & Saito, T., 1996. Designing Interface Toolkit with Dynamic Selectable Modality. In *Proceedings of the 2nd Annual ACM Conference on Assistive Technologies*. Vancouver, British Columbia, Canada: ACM, pp. 72–79. <http://dx.doi.org/10.1145/228347.228360>.
- (**Kay 1993**) Kay, R.H., 1993. A Practical Research Tool for Assessing Ability to Use Computers: The Computer Ability Survey (CAS). *Journal of Research on Computing in Education*, 26(1), pp.16–27. issn:0888-6504.

- (**Keates et al. 2000**) Keates, S. et al., 2000. Towards a Practical Inclusive Design Approach. In *Proceedings on the 2000 Conference on Universal Usability*. Arlington, Virginia, USA: ACM, pp. 45–52. <http://dx.doi.org/10.1145/355460.355471>.
- (**Kent 2002**) Kent, S., 2002. Model Driven Engineering. In *Proceedings of the 3rd International Conference on Integrated Formal Methods*. Turku, Finland: Springer, pp. 286–298. http://dx.doi.org/10.1007/3-540-47884-1_16.
- (**Kiczales et al. 1997**) Kiczales, G. et al., 1997. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. European Conference on Object-Oriented Programming. pp. 220–242. <http://dx.doi.org/10.1007/BFb0053381>.
- (**Kramer & Magee 2007**) Kramer, J. & Magee, J., 2007. Self-Managed Systems: an Architectural Challenge. In *Proceedings of the Workshop on the Future of Software Engineering*. International Conference on Software Engineering. Minneapolis, USA: IEEE, pp. 259–268. <http://dx.doi.org/10.1109/FOSE.2007.19>.
- (**Krasner & Pope 1988**) Krasner, G.E. & Pope, S.T., 1988. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object Oriented Programming*, 1(3), pp.26–49.
- (**Lafreniere et al. 2011**) Lafreniere, B. et al., 2011. AdaptableGIMP: Designing a Socially-Adaptable Interface. In *Proceedings of the 24th ACM Symposium Adjunct on User Interface Software and Technology*. Santa Barbara, California, USA: ACM, pp. 89–90. <http://dx.doi.org/10.1145/2046396.2046437>.
- (**Landauer 1997**) Landauer, T.K., 1997. Behavioral Research Methods in Human-Computer Interaction. In M. G. Helander, T. K. Landauer, & P. V. Prabhu, eds. *Handbook of Human-Computer Interaction*. Amsterdam: Elsevier, pp. 203–227. <http://dx.doi.org/10.1023/A:1008171820407>.
- (**Lecolinet 2003**) Lecolinet, E., 2003. A Molecular Architecture for Creating Advanced GUIs. In *Proceedings of the 16th ACM Symposium on User Interface Software and Technology*. Vancouver, Canada: ACM, pp. 135–144. <http://dx.doi.org/10.1145/964696.964711>.
- (**Lehmann et al. 2010**) Lehmann, G. et al., 2010. A 3-Layer Architecture for Smart Environment Models. In *Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications*. Mannheim, Germany: IEEE, pp. 636–641. <http://dx.doi.org/10.1109/PERCOMW.2010.5470513>.
- (**Leon 2008**) Leon, A., 2008. *ERP Demystified* 2nd ed., New Delhi: Tata McGraw-Hill. isbn:9780070656642.
- (**Lepreux et al. 2007**) Lepreux, S., Vanderdonckt, J. & Michotte, B., 2007. Visual Design of User Interfaces by (De)Composition. In *Proceedings of the 13th International Conference on Interactive Systems: Design, Specification,*

- and Verification*. Dublin, Ireland: Springer, pp. 157–170.
http://dx.doi.org/10.1007/978-3-540-69554-7_13.
- (Limbourg et al. 2004)** Limbourg, Q. et al., 2004. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *Engineering Advanced Web Applications: Proceedings of Workshops in connection with the 4th International Conference on Web Engineering*. Munich, Germany: Rinton Press, pp. 325–338. isbn:1-58949-046-0.
- (Lin & Landay 2008)** Lin, J. & Landay, J.A., 2008. Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. Florence, Italy: ACM, pp. 1313–1322.
<http://dx.doi.org/10.1145/1357054.1357260>.
- (Lonczewski & Schreiber 1996)** Lonczewski, F. & Schreiber, S., 1996. The FUSE-System: an Integrated User Interface Design Environment. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium: Springer, pp. 37–56. isbn:2-87037-232-9.
- (López-Jaquero et al. 2009)** López-Jaquero, V., Montero, F. & Real, F., 2009. Designing User Interface Adaptation Rules with T:XML. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*. Sanibel Island, Florida, USA: ACM, pp. 383–388.
<http://dx.doi.org/10.1145/1502650.1502705>.
- (Lykkegaard & Elbak 2011)** Lykkegaard, B. & Elbak, A., 2011. *IDC - Document at a Glance - LC52T*, International Data Corporation (IDC).
<http://bit.ly/IDC-LC52T> [Accessed May 2, 2012].
- (Mace et al. 1990)** Mace, R.L., Hardie, G. e J. & Place, J.P., 1990. Accessible Environments: Toward Universal Design. In *Design Intervention: Toward a More Humane Architecture*. Center for Accessible Housing, North Carolina State University. isbn:978-0442273330.
- (Markopoulos et al. 1992)** Markopoulos, P. et al., 1992. Adept-A task based design environment. In *Proceedings of the 25th Hawaii International Conference on System Sciences*. Hawaii, USA: IEEE, pp. 587–596.
<http://dx.doi.org/10.1109/HICSS.1992.183310>.
- (Märting 1996)** Märting, C., 1996. Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In *Proceedings of the 2nd International on Computer-Aided Design of User Interfaces*. Namur, Belgium: Presses Universitaires de Namur, pp. 57–76. <http://bit.ly/1lCPVEm>.
- (Martin Fowler 2006)** Martin Fowler, 2006. GUI Architectures.
<http://bit.ly/1fwn70o> [Accessed May 1, 2014].
- (Mayhew 1999)** Mayhew, D.J., 1999. The Usability Engineering Lifecycle. In *Proceedings of the Extended Abstracts of the 17th Conference on Human Factors in Computing Systems*. Pittsburgh, Pennsylvania: ACM, pp. 147–148.
<http://dx.doi.org/10.1145/632716.632805>.

- (**McGrenere 2000**) McGrenere, J., 2000. "Bloat": The Objective and Subject Dimensions. In *Proceedings of the Extended Abstracts of the 18th Conference on Human Factors in Computing Systems*. The Hague, The Netherlands: ACM, pp. 337–338. <http://dx.doi.org/10.1145/633292.633495>.
- (**McGrenere et al. 2007**) McGrenere, J., Baecker, R.M. & Booth, K.S., 2007. A Field Evaluation of an Adaptable Two-Interface Design for Feature-Rich Software. *ACM Transactions on Computer-Human Interaction*, 14(1), pp.1–43. <http://dx.doi.org/10.1145/1229855.1229858>.
- (**McGrenere et al. 2002**) McGrenere, J., Baecker, R.M. & Booth, K.S., 2002. An Evaluation of a Multiple Interface Design Solution for Bloated Software. In *Proceedings of the 20th SIGCHI Conference on Human Factors in Computing Systems*. Minneapolis, Minnesota, USA: ACM, pp. 164–170. <http://dx.doi.org/10.1145/503376.503406>.
- (**Meixner et al. 2011**) Meixner, G., Paternò, F. & Vanderdonckt, J., 2011. Past, Present, and Future of Model-Based User Interface Development. *i-com*, 10(3), pp.2–11. <http://dx.doi.org/10.1524/icom.2011.0026>.
- (**Meskens et al. 2008**) Meskens, J. et al., 2008. Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me. In *Proceedings of the 8th Working Conference on Advanced Visual Interfaces*. Napoli, Italy: ACM, pp. 233–240. <http://dx.doi.org/10.1145/1385569.1385607>.
- (**Michotte & Vanderdonckt 2008**) Michotte, B. & Vanderdonckt, J., 2008. GrafiXML, a Multi-target User Interface Builder Based on UsiXML. In *Proceedings of the 4th International Conference on Autonomic and Autonomous Systems*. Cancun, Mexico: IEEE, pp. 15 –22. <http://dx.doi.org/10.1109/ICAS.2008.29>.
- (**Microsoft 2011**) Microsoft, 2011. Role based UI - Dynamics CRM 2011. <http://bit.ly/DynamicsRoleBasedUI> [Accessed August 31, 2012].
- (**Montero & López-Jaquero 2007**) Montero, F. & López-Jaquero, V., 2007. IdealXML: An Interaction Design Tool. In G. Calvary et al., eds. *Computer-Aided Design of User Interfaces V*. Springer, pp. 245–252. isbn:978-1-4020-5819-6, 978-1-4020-5820-2.
- (**Mori et al. 2002**) Mori, G., Paternò, F. & Santoro, C., 2002. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering*, 28(8), pp.797–813. <http://dx.doi.org/10.1109/TSE.2002.1027801>.
- (**Myers et al. 2000**) Myers, B., Hudson, S.E. & Pausch, R., 2000. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 7(1), pp.3–28. <http://dx.doi.org/10.1145/344949.344959>.
- (**Myers et al. 1997**) Myers, B.A. et al., 1997. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE*

- Transactions on Software Engineering*, 23(6), pp.347–365.
<http://dx.doi.org/10.1109/32.601073>.
- (**Nebeling et al. 2012**) Nebeling, M., Leone, S. & Norrie, M., 2012. Crowdsourced Web Engineering and Design. In M. Brambilla, T. Tokuda, & R. Tolksdorf, eds. *Web Engineering*. Springer, pp. 31–45. isbn:978-3-642-31752-1.
- (**Nebeling & Norrie 2011**) Nebeling, M. & Norrie, M.C., 2011. Tools and Architectural Support for Crowdsourced Adaptation of Web Interfaces. In *Proceedings of the 11th International Conference on Web Engineering*. Paphos, Cyprus: Springer, pp. 243–257. isbn:978-3-642-22232-0.
- (**Nichols et al. 2004**) Nichols, J., Myers, B.A. & Litwack, K., 2004. Improving Automatic Interface Generation with Smart Templates. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*. Funchal, Madeira, Portugal: ACM, pp. 286–288. <http://dx.doi.org/10.1145/964442.964507>.
- (**Norcio & Stanley 1989**) Norcio, A.F. & Stanley, J., 1989. Adaptive Human-Computer Interfaces: A Literature Survey and Perspective. *IEEE Transactions on Systems, Man, and Cybernetics*, 19, pp.399–408. <http://dx.doi.org/10.1109/21.31042>.
- (**Nylander et al. 2004**) Nylander, S., Bylund, M. & Wærn, A., 2004. The Ubiquitous Interactor - Device Independent Access to Mobile Services. In *Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces*. Funchal, Portugal: Kluwer, pp. 269–280. http://dx.doi.org/10.1007/1-4020-3304-4_22.
- (**Olsen,Jr. 1989**) Olsen,Jr., D.R., 1989. A Programming Language Basis for User Interface Management. In *Proceedings of the 7th ACM SIGCHI Conference on Human Factors in Computing Systems*. Austin, USA: ACM, pp. 171–176. <http://dx.doi.org/10.1145/67449.67485>.
- (**Olsen,Jr. 2007**) Olsen,Jr., D.R., 2007. Evaluating User Interface Systems Research. In *Proceedings of the 20th ACM SIGCHI Symposium on User Interface Software and Technology*. Newport, Rhode Island, USA: ACM, pp. 251–258. <http://dx.doi.org/10.1145/1294211.1294256>.
- (**Olsen,Jr. 1986**) Olsen,Jr., D.R., 1986. MIKE: The Menu Interaction Kontrol Environment. *ACM Transactions on Graphics*, 5(4), pp.318–344. <http://dx.doi.org/10.1145/27623.28868>.
- (**OMG 2013**) OMG, 2013. Object Management Group - Model-Driven Architecture. <http://www.omg.org/mda> [Accessed April 10, 2013].
- (**Oreizy et al. 1999**) Oreizy, P. et al., 1999. An Architecture-Based Approach to Self-Adaptive Software. *Intelligent Systems and Their Applications, IEEE*, 14(3), pp.54–62. <http://dx.doi.org/10.1109/5254.769885>.

- (Oz 2008) Oz, E., 2008. *Management Information Systems, Sixth Edition* 6th ed., Boston, MA, United States: Course Technology Press. isbn:1423901789, 9781423901785.
- (Palay et al. 1989) Palay, A.J. et al., 1989. The Andrew Toolkit: An Overview. In *Proceedings of the 1988 Winter USENIX Technical Conference*. Dallas, Texas: USENIX Association, pp. 9–21. <http://bit.ly/SdJ4H5>.
- (Panorama Consulting Group 2010) Panorama Consulting Group, 2010. A Guide to Increasing User Acceptance of ERP Systems. <http://bit.ly/R6QWJi>.
- (Paolacci et al. 2010) Paolacci, G., Chandler, J. & Ipeirotis, P.G., 2010. Running Experiments on Amazon Mechanical Turk. *Judgment and Decision Making*, 5(5), pp.411–419.
- (Paternò 1999) Paternò, F., 1999. *Model-based Design and Evaluation of Interactive Applications* 1st ed., London, UK: Springer. isbn:1852331550.
- (Paternò et al. 1997) Paternò, F., Mancini, C. & Meniconi, S., 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the 6th International Conference on Human-Computer Interaction*. Sydney, Australia: Chapman and Hall, pp. 362–369. http://dx.doi.org/10.1007/978-0-387-35175-9_58.
- (Paterno' et al. 2009) Paterno', F., Santoro, C. & Spano, L.D., 2009. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *ACM Transactions on Computer-Human Interaction*, 16(4), pp.19:1–19:30. <http://dx.doi.org/10.1145/1614390.1614394>.
- (Peissner et al. 2012) Peissner, M. et al., 2012. MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark: ACM, pp. 81–90. <http://dx.doi.org/10.1145/2305484.2305500>.
- (Pérez-Medina et al. 2007) Pérez-Medina, J.-L., Dupuy-Chessa, S. & Front, A., 2007. A Survey of Model Driven Engineering Tools for User Interface Design. In M. Winckler, H. Johnson, & P. Palanque, eds. *Task Models and Diagrams for User Interface Design*. Springer, pp. 84–97. isbn:978-3-540-77221-7.
- (Piechnick et al. 2012) Piechnick, C. et al., 2012. Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation – Smart Application Grids. In *Proceedings of the 4th International Conference on Adaptive and Self-adaptive Systems and Applications*. Nice, France: IARIA, pp. 93–102. isbn:978-1-61208-219-6.
- (Pleuss et al. 2010) Pleuss, A., Botterweck, G. & Dhungana, D., 2010. Integrating Automated Product Derivation and Individual User Interface Design. In *Proceedings of the 4th International Workshop on Variability*

- Modelling of Software-Intensive Systems*. Linz, Austria: Universitat Duisburg-Essen, pp. 69–76. <http://bit.ly/1nM3Vhh>.
- (**Puerta & Eisenstein 1998**) Puerta, A. & Eisenstein, J., 1998. Interactively Mapping Task Models to Interfaces in MOBI-D. In *Proceedings of Eurographics Workshop on Design, Specification and Validation of Interactive Systems*. Abingdon, United Kingdom: Springer, pp. 261–273. <http://b.gatech.edu/1lRy0Kg>.
- (**Puerta & Eisenstein 2002**) Puerta, A. & Eisenstein, J., 2002. XIML: A Common Representation for Interaction Data. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*. San Francisco, California, USA: ACM, pp. 214–215. <http://dx.doi.org/10.1145/502716.502763>.
- (**Puerta 1996**) Puerta, A.R., 1996. The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. In *Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces*. Namur, Belgium: Presses Universitaires de Namur, pp. 19–36. <http://bit.ly/1u8ZWNw>.
- (**Raneburger et al. 2012**) Raneburger, D., Popp, R. & Vanderdonckt, J., 2012. An Automated Layout Approach for Model-Driven WIMP-UI Generation. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark: ACM, pp. 91–100. <http://dx.doi.org/10.1145/2305484.2305501>.
- (**Reinecke & Bernstein 2011**) Reinecke, K. & Bernstein, A., 2011. Improving Performance, Perceived Usability, and Aesthetics with Culturally Adaptive User Interfaces. *ACM Transactions on Computer-Human Interaction*, 18(2), pp.1–29. <http://dx.doi.org/10.1145/1970378.1970382>.
- (**Salehie & Tahvildari 2009**) Salehie, M. & Tahvildari, L., 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2), pp.1–42. <http://dx.doi.org/10.1145/1516533.1516538>.
- (**Schmucker 1987**) Schmucker, K.J., 1987. MacApp: An Application Framework. In R. M. Baecker & W. A. S. Buxton, eds. *Human-computer Interaction*. Morgan Kaufmann Publishers Inc., pp. 591–594. isbn:0-934613-24-9.
- (**Schwartz et al. 2009**) Schwartz, V., Feuerstack, S. & Albayrak, S., 2009. Behavior-Sensitive User Interfaces for Smart Environments. In *Proceedings of the 2nd International Conference on Digital Human Modeling*. San Diego, USA: Springer, pp. 305–314. isbn:978-3-642-02808-3.
- (**Sharp et al. 2007**) Sharp, H., Rogers, Y. & Preece, J., 2007. *Interaction Design: Beyond Human-Computer Interaction* 2nd ed., Wiley. isbn:0470018666.

- (**Shaw 2002**) Shaw, M., 2002. What Makes Good Research in Software Engineering. *International Journal on Software Tools for Technology Transfer*, 4(1), pp.1–7. <http://dx.doi.org/10.1007/s10009-002-0083-4>.
- (**Shneiderman 2003**) Shneiderman, B., 2003. Promoting Universal Usability with Multi-Layer Interface Design. In *Proceedings of the Conference on Universal Usability*. Vancouver, Canada: ACM, pp. 1–8. <http://dx.doi.org/10.1145/957205.957206>.
- (**Da Silva 2001**) Da Silva, P.P., 2001. User Interface Declarative Models and Development Environments: A Survey. In *Proceedings of the 7th International Conference on Design, Specification, and Verification of Interactive Systems*. Limerick, Ireland: Springer, pp. 207–226. http://dx.doi.org/10.1007/3-540-44675-3_13.
- (**Singh & Wesson 2009**) Singh, A. & Wesson, J., 2009. Evaluation Criteria for Assessing the Usability of ERP Systems. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. Emfuleni, South Africa: ACM, pp. 87–95. <http://dx.doi.org/10.1145/1632149.1632162>.
- (**Soley & OMG Staff Strategy Group 2000**) Soley, R. & OMG Staff Strategy Group, 2000. Model Driven Architecture. <http://bit.ly/ModelDrivenArch> [Accessed April 10, 2013].
- (**Stephanidis 1997**) Stephanidis, C., 1997. Towards the Next Generation of UIST: Developing for all Users. In *Proceedings of the 7th International Conference on Human-Computer Interaction*. New York, NY, USA: Elsevier Science Inc., pp. 473–476. isbn:0-444-82183-X.
- (**Stuerzlinger et al. 2006**) Stuerzlinger, W. et al., 2006. User Interface Façades: Towards Fully Adaptable User Interfaces. In *Proceedings of the 19th ACM Symposium on User Interface Software and Technology*. Montreux, Switzerland: ACM, pp. 309–318. <http://dx.doi.org/10.1145/1166253.1166301>.
- (**Synactive GmbH 2010**) Synactive GmbH, 2010. GuiXT - Simplify and Optimize the SAP ERP User Interface. <http://bit.ly/SAPGuiXTSimplifyUI> [Accessed September 4, 2012].
- (**Szekely et al. 1995**) Szekely, P. et al., 1995. Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach. In *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*. Yellowstone Park, Wyoming, USA: Chapman & Hall, Ltd., pp. 120–150. isbn:0-412-72180-5.
- (**Szekely et al. 1992**) Szekely, P., Luo, P. & Neches, R., 1992. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings of the 10th ACM SIGCHI Conference on Human Factors in Computing Systems*. Monterey, California, USA: ACM, pp. 507–515. <http://dx.doi.org/10.1145/142750.142912>.

- (**Topi et al. 2005**) Topi, H., Lucas, W.T. & Babaian, T., 2005. Identifying Usability Issues with an ERP Implementation. In *Proceedings of the 7th International Conference on Enterprise Information Systems*. Miami, USA: SciTePress, pp. 128–133. <http://bit.ly/1kxcARb>.
- (**Uflacker & Busse 2007**) Uflacker, M. & Busse, D., 2007. Complexity in Enterprise Applications vs. Simplicity in User Experience. In *Proceedings of the 12th International Conference on Human-Computer Interaction: Applications and Services*. Beijing, China: Springer, pp. 778–787. isbn:978-3-540-73109-2.
- (**Vanderdonckt 2008**) Vanderdonckt, J., 2008. Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges. *Romanian Journal of Human - Computer Interaction*, 1(1), pp.1–10. issn:1843-4460.
- (**Vanderdonckt & Bodart 1996**) Vanderdonckt, J. & Bodart, F., 1996. The “Corpus Ergonomicus”: A Comprehensive and Unique Source for Human-Machine Interface. In *Proceedings of the 1st International Conference on Applied Ergonomics*. Istanbul, Turkey: USA Publishing, pp. 162–169. <http://bit.ly/1jljedM>.
- (**Vanderdonckt & Bodart 1993**) Vanderdonckt, J.M. & Bodart, F., 1993. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands: ACM, pp. 424–429. <http://dx.doi.org/10.1145/169059.169340>.
- (**Wiecha et al. 1990**) Wiecha, C. et al., 1990. ITS: A Tool for Rapidly Developing Interactive Applications. *ACM Transactions on Information Systems*, 8(3), pp.204–236. <http://dx.doi.org/10.1145/98188.98194>.

Appendices

A

Algorithms

This appendix includes the complete pseudo-code and complexity analysis of algorithms which were presented in the thesis as excerpts.

A.1 Feature-Set Minimization Algorithm

Variables: m = Number of Task Models, n = Number of Tasks in a Task Model, j = Number of User Roles, k = Number of Blocked CUI Elements for a given Task, p = Number of Parent Tasks for a given Task, l = Number of Task Roles

Legend: **CON** = Constant, **LOG** = Logarithmic, **POL** = Polynomial

Total Running Time: $O(m \times (n \times l \times p \times (2 \times j \times \log j + k) + n))$

	Type	Cost	Time	Pseudo-code
				$O(m \times (n \times l \times p \times (2 \times j \times \log j + k) + n))$
1.	.	.	.	Minimize-Feature-Set (UserRef, UserInterfaceID) {
2.	.	.	.	//Load UI Related Information From Database
3.	CON	c1	$O(1)$	UserRoles[] \leftarrow GetUserRolesFromDB(UserRef)
4.	CON	c2	$O(1)$	UIModel \leftarrow GetUIModelFromDB(UserInterfaceID)
5.	.	.	.	//Simplify All Associated Task Models
6.	POL	c3	$O(m)$	foreach tm in UIModel.TaskModel[] {
7.	.	.	.	Simplify-Task-Model(tm.TaskModelID,
8.	.	.	.	UserRoles, UIModel)
9.	.	.	.	}
10.	.	.	.	}

$$O(n \times l \times p \times (2 \times j \times \log j + k) + n)$$

```

1.  .      .      .      Simplify-Task-Model (TaskModelID, UserRoles[],
2.  .      .      .      UIModel, TasksUnblockedByTheUser[])
3.  .      .      .      {
4.  .      .      .      //Get Task Model's Tasks, Mappings, and CUI
5.  CON    c1      0 (1)    Tasks[] ← Select t From UIModel.Task
6.  .      c1      .      Where t.TaskModelID = TaskModelID
7.  .      .      .
8.  CON    c2      0 (1)    TMTToAUIMapping[] ← Select m From
9.  .      .      .      UIModel.MappingTaskModelToAUI[]
10. .      c2      .      Where m.TaskModelID = TaskModelID
11. .      .      .
12. CON    c3      0 (1)    AUIToCUIMapping[] ← Select m From
13. .      c3      .      UIModel.MappingAUIToCUI[] Where
14. .      c3      .      TMTToAUIMapping.Contains(m.AUIModelID) = true
15. .      .      .
16. CON    c4      0 (1)    CUIElements[] ← Select c From
17. .      .      .      UIModel.CUIElements[] Where
18. .      c4      .      AUIToCUIMapping.Contains(c.CUIModelID) = true
19. .      .      .
20. .      .      .      //Simplify All the Task Model's Tasks
21. POL    c5      0 (n × l × p) foreach task in Tasks[] {
22. .      .      .      if TasksUnblockedByTheUser.Contains(
23. .      .      .      task) = false
24. .      .      .      TaskRoles[] ← Get-Task-Roles(
25. .      .      .      task, UIModel)
26. .      .      .      Simplify-Task(task.TaskID, UserRoles,
27. .      .      .      TaskRoles, TMTToAUIMapping,
28. .      .      .      AUIToCUIMapping, CUIElements)
29. .      .      .      }
30. .      .      .      //Re-enabled Read-Only Containers with
31. .      .      .      //Non-Read-Only Children
32. CON    c6      0 (1)    DisabledCUIContainers[] ←
33. .      .      .      Select c From UIModel.CUIElements[]
34. .      c6      .      Where c.ReadOnly = true and
35. .      c6      .      (Select child From UIModel.CUIElements[]
36. .      c6      .      Where child.ReadOnly = false

```

```

37. .      c6      .      and child.ParentElementID =
38. .      c6      .      c.ElementID).Count() > 0
39. .      .      .
40. POL    c7      O(n)    foreach cuiEl in DisabledCUIContainers
41. .      c7      .      { cuiEl.IsReadOnly = false }
42. .      .      .      }

```

$O(2 \times j \times \log j + k)$

```

1. .      .      .      Simplify-Task (TaskID, UserRoles[],
2. .      .      .      TaskRoles[], TMTToAUIMapping[],
3. .      .      .      AUIToCUIMapping[], CUIElements[])
4. .      .      .      {
5. .      .      .      //Order Assigned Roles by Task Role Priority
6. POL    c1      O(j)    foreach ur in UserRoles {
7. CON    c2      O(1)    tr ← Select From TaskRoles
8. .      c2      O(1)    Where t.RoleRef = ur.RoleRef
9. .      .      .
10. CON    c3      O(1)    if tr = null {
11. CON    c4      O(1)    tr ← Select t From TaskRoles
12. .      c4      .      Where t.RoleRef = All-Roles
13. CON    c5      O(1)    ur.Priority ← tr.Priority;
14. .      .      .      }
15. .      .      .
16. LOG    c6      O(j × log j) UserRoles.OrderBy(Priority)
17. CON    c7      O(1)    primaryRole ← UserRoles.First()
18. CON    c8      O(1)    if primaryRole.RoleRef ≠ All-Roles {
19. .      .      .      //Simplify CUI
20. .      .      .      Simplify-CUI(PrimaryRole, TaskID,
21. .      .      .      TMTToAUIMapping, AUIToCUIMapping,
22. .      .      .      CUIElements)
23. .      .      .      }
24. .      .      .      }

```

$O(k)$

```

1.  .      .      .      Simplify-CUI (PrimaryRole, TaskID,
2.  .      .      .      TMTToAUIMapping[], AUIToCUIMapping[],
3.  .      .      .      CUIElements[])
4.  .      .      .      {
5.  CON    c1      0 (1)    blockedAUIElementIDs[] ← Select
6.  .      c1      .        el.AUIElementID From TMTToAUIMapping
7.  .      c1      .        Where el.TaskID = TaskID
8.  .      .      .
9.  CON    c2      0 (1)    blockedCUIElementIDs[] ← Select
10. .      c2      .        el.CUIElementID From AUIToCUIMapping[]
11. .      c2      .        Where blockedAUIElementIDs.Contains(
12. .      c2      .        el.AUIElementID) = true
13. .      .      .
14. CON    c3      0 (1)    blockedCUIElements[] ← Select el
15. .      c3      .        From CUIElements Where
16. .      c3      .        blockedCUIElementIDs.Contains(
17. .      c3      .        el.WidgetID) = true
18. .      .      .
19. .      .      .        //Apply Concrete Operation to CUI
20. POL    c4      0 (k)    foreach element in blockedCUIElements {
21. POL    c5      0 (1)    switch PrimaryRole.ConcreteOperation
22. .      c5      .        case Hide
23. CON    c6      0 (1)    element.Visible ← false; break;
24. .      c5      .        case Disable
25. CON    c7      0 (1)    element.ReadOnly ← true; break;
26. .      c5      .        case Protect
27. CON    c8      0 (1)    element.ReadOnly ← true;
28. CON    c9      0 (1)    element.MaskChar ← '*'; break;
29. .      c5      .        case Fade
30. CON    c10     0 (1)    element.Opacity ← 30%; break;
31. .      .      .        }
32. .      .      .      }

```

$O(l \times p)$

```

1. . . . [] Get-Task-Roles (Task, UIModel)
2. . . . {
3. CON c1  $O(1)$  TaskRoles[] ← Select tr From
4. . . . UIModel.TaskRoles Where
5. . . . tr.TaskID = Task.TaskID
6. . . .
7. CON c2  $O(1)$  if Task.ParentRoleInheritance = Merge {
8. CON c3  $O(1)$  ParentTask ← (Select t From
9. . . . UIModel.Task[] Where
10. . . . t.TaskID = Task.ParentTaskID).First()
11. . . .
12. CON c4  $O(1)$  if ParentTask = null {return TaskRoles;}
13. . . .
14. POL c5  $O(l \times p)$  TaskRoles.Merge(Get-Task-Roles(
15. . . . ParentTask, UIModel))
16. . . . }
17. CON c6  $O(1)$  return TaskRoles
18. . . . }

```

A.2 Layout Optimization Algorithm

Variables: **m** = Number of User Roles, **n** = Number of Workflows Assigned to the Primary Role

Legend: **CON** = Constant, **LOG** = Logarithmic, **POL** = Polynomial

Total Running Time: $O(2 \times m \times \log m + 2 \times n \times \log n)$

	Type	Cost	Time	Pseudo-code
1.	.	.	.	Optimize-Layout (UserRoles[], Roles[],
2.	.	.	.	UIModel, LayoutID, WorkflowsCancelledByUser[],
3.	.	.	.	AlternativeWorkflows[]) {
4.	.	.	.	//Order Assigned Roles by Role Priorities
5.	POL	c1	$O(m)$	foreach ur in UserRoles {
6.	CON	c2	$O(1)$	tr ← Roles[].GetRole(ur.RoleRef)
7.	CON	c3	$O(1)$	if tr = null
8.	CON	c4	$O(1)$	tr ← Roles[].GetRole(All-Roles)
9.	CON	c5	$O(1)$	ur.Priority ← tr.Priority;
10.	.	.	.	}
11.	.	.	.	
12.	LOG	c6	$O(m \times \log m)$	UserRoles.OrderBy(Priority)
13.	CON	c7	$O(1)$	primaryRole ← UserRoles.First()
14.	.	.	.	
15.	CON	c8	$O(1)$	Workflows[] ← Get-Workflows(primaryRole,
16.	.	c8	.	LayoutID, AlternativeWorkflows)
17.	CON	c9	$O(1)$	Workflows = Select w From Workflows Where
18.	.	c9	.	WorkflowsCancelledByUser.Contains(w) = false
19.	.	.	.	
20.	CON	c10	$O(n \times \log n)$	Workflows.OrderBy(ExecutionOrder)
21.	CON	c11	$O(n)$	foreach workflow in Workflows[] {
22.	.	.	.	//time depends on workflow's content
23.	.	.	.	workflow.Execute(UIModel)
24.	.	.	.	}
25.	.	.	.	}

A.3 Conflict Checking Based on Temporal Constraints

Variables: m = number of unselected tasks, n = number of conflicting tasks

Legend: CON = Constant, POL = Polynomial

Total Running Time: $O(m)$

	Type	Cost	Time	Pseudo-code
1.				[] CheckForConflicts(TaskModel TM) {
2.				//Get unselected tasks and their relationships
3.	CON	c1	$O(1)$	UnselectedTasks[] \leftarrow Select * From TM.Tasks
4.	.	c1	$O(1)$	Where Selected = false
5.	CON	c2	$O(1)$	UnselTaskRelationships[] \leftarrow Select * From
6.	.	c2	$O(1)$	TM.Relationships as R Where (Select TaskID
7.	.	c2	$O(1)$	From UnselectedTasks).Contains(
8.	.	c2	$O(1)$	R.SourceTaskID) (Select TaskID From
9.	.	c2	$O(1)$	UnselectedTasks).Contains(R.TargetTaskID)
10.	.	.	.	//CTT relation types indicating dependency
11.	CON	c3	$O(1)$	RemoveTAIfTBIsRemoved[] \leftarrow {
12.	.	c3	$O(1)$	Concurrency with Info. Exchange }
13.	.	c4	$O(1)$	RemoveTBIfTAIsRemoved[] \leftarrow {
14.	.	c4	$O(1)$	Concurrency with Info. Exchange,
15.	.	c4	$O(1)$	Enabling, Enabling with Info. Exchange }
16.	CON	c5	$O(1)$	ConflictingTasks \leftarrow [];
17.	POL	c6	$O(m)$	foreach uTask in UnselectedTasks {
18.	.	.	.	//Get conflicts created by unselecting task
19.	CON	c7	$O(1)$	ConflictingTasks.Add(Select * From
20.	.	c7	$O(1)$	TM.Tasks as T Where (Select SourceTaskID
21.	.	c7	$O(1)$	From UnselTaskRelationships Where
22.	.	c7	$O(1)$	TargetTaskID = uTask.TaskID &&
23.	.	c7	$O(1)$	RemoveTAIfTBIsRemoved.Contains(
24.	.	c7	$O(1)$	RelType)).Contains(T.TaskID) (Select
25.	.	c7	$O(1)$	TargetTaskID From UnselTaskRelationships
26.	.	c7	$O(1)$	Where SourceTaskID = uTask.TaskID &&
27.	.	c7	$O(1)$	RemoveTBIfTAIsRemoved.Contains(
28.	.	c7	$O(1)$	RelType)).Contains(T.TaskID)) }
29.	CON	c8	$O(1)$	return ConflictingTasks }

B

Questionnaires

This appendix presents the questionnaires, which were used in the usability studies that were conducted to evaluate our UI adaptation mechanism.

B.1 Demographics Questions Used in Usability Studies

1) Please indicate your gender.

☐ Male ☐ Female

2) Please indicate your age group.

☐ Less than 21 ☐ 21 to 30 ☐ 31 to 40 ☐ 41 to 50
☐ 51 to 60 ☐ Above 60

3) Please indicate the highest degree you have obtained.

☐ Primary School ☐ High School ☐ Bachelor Degree
☐ Master Degree ☐ Doctorate / PhD

4) Have you ever used enterprise software applications or any line-of-business information system (enterprise resource planning, customer relationship management, marketing information management, supply chain management, accounting, education information management, etc.)?

☐ Yes ☐ No

5) If your answer to question 4 is 'Yes', for how long did you use or have been using such systems?

- ☐ Less than 1 Year ☐ 1 to 5 Years ☐ 6 to 10 Years
☐ 11 to 15 Years ☐ 16 to 20 Years ☐ Over 20 Years

6) If your answer to question 4 is 'Yes', how many hours per day did you use or have been using such systems?

- ☐ Less than 1 Hour ☐ 1 to 4 Hours
☐ 5 to 8 Hours ☐ Over 8 Hours

7) How would you rate your computer skills?

Bad ☆ ☆ ☆ ☆ ☆ ☆ ☆ Good

How would you rate your ability to:

8) Use a word processor to create documents

Low ☆ ☆ ☆ ☆ ☆ ☆ ☆ High

9) Learn a software package that you never used before

Low ☆ ☆ ☆ ☆ ☆ ☆ ☆ High

10) Use an operating system (Windows, Mac OS, Linux, etc.)

Low ☆ ☆ ☆ ☆ ☆ ☆ ☆ High

11) Discuss strengths and weaknesses of various software packages

Low ☆ ☆ ☆ ☆ ☆ ☆ ☆ High

To what extent would you agree with the following?

12) I could probably teach myself most of the things I need to know about computers.

Disagree ☆ ☆ ☆ ☆ ☆ ☆ ☆ Agree

13) If I had a problem using the computer, I could solve it one way or another.

Disagree ☆ ☆ ☆ ☆ ☆ ☆ ☆ Agree

14) I do not need someone to tell me the best way to use a computer.

Disagree ☆ ☆ ☆ ☆ ☆ ☆ ☆ Agree

15) I prefer to learn a new computer software package on my own.

Disagree ☆ ☆ ☆ ☆ ☆ ☆ ☆ Agree

The computer literacy level of the participants is calculated using the answers they gave to questions 7 to 15. The average of the answers that were given to questions 8 to 15 is computed. Then, the average between the result and the answer given to question 7 is calculated. The following equation shows the calculation:

$$\frac{(\sum_{n=8}^{15} AnswerToQuestion_n)/8 + AnswerToQuestion_7}{2}$$

The rating (1 to 7) calculated by the equation above determines the computer literacy level as follows: novice (1, 2, 3), intermediate (4, 5), and expert (6, 7).

B.2 System Usability Scale (SUS)

Please answer the questions below to rate the user interface based on its ability to support the given task.

	Strongly Disagree	Strongly Agree
1) I think that I would like to use this system frequently	<input type="radio"/>	<input type="radio"/>
2) I found the system unnecessarily complex	<input type="radio"/>	<input type="radio"/>
3) I thought the system was easy to use	<input type="radio"/>	<input type="radio"/>
4) I think that I would need the support of a technical person to be able to use this system	<input type="radio"/>	<input type="radio"/>
5) I found the various functions in this system were well integrated	<input type="radio"/>	<input type="radio"/>
6) I thought there was too much inconsistency in this system	<input type="radio"/>	<input type="radio"/>
7) I would imagine that most people would learn to use this system very quickly	<input type="radio"/>	<input type="radio"/>
8) I found the system very cumbersome to use	<input type="radio"/>	<input type="radio"/>
9) I felt very confident using the system	<input type="radio"/>	<input type="radio"/>
10) I needed to learn a lot of things before I could get going with this system	<input type="radio"/>	<input type="radio"/>

Any other comments: _____

Note: This questionnaire was created by Brooke (1996).

B.3 Microsoft Product Reaction Cards

Please select three of the following terms that you find the most suitable for describing the user interface.

<input type="checkbox"/> Accessible	<input type="checkbox"/> Advanced	<input type="checkbox"/> Annoying	<input type="checkbox"/> Appealing
<input type="checkbox"/> Approachable	<input type="checkbox"/> Attractive	<input type="checkbox"/> Boring	<input type="checkbox"/> Business-like
<input type="checkbox"/> Busy	<input type="checkbox"/> Calm	<input type="checkbox"/> Clean	<input type="checkbox"/> Clear
<input type="checkbox"/> Collaborative	<input type="checkbox"/> Comfortable	<input type="checkbox"/> Compatible	<input type="checkbox"/> Compelling
<input type="checkbox"/> Complex	<input type="checkbox"/> Comprehensive	<input type="checkbox"/> Confident	<input type="checkbox"/> Confusing
<input type="checkbox"/> Connected	<input type="checkbox"/> Consistent	<input type="checkbox"/> Controllable	<input type="checkbox"/> Convenient
<input type="checkbox"/> Creative	<input type="checkbox"/> Customizable	<input type="checkbox"/> Cutting edge	<input type="checkbox"/> Dated
<input type="checkbox"/> Desirable	<input type="checkbox"/> Difficult	<input type="checkbox"/> Disconnected	<input type="checkbox"/> Disruptive
<input type="checkbox"/> Distracting	<input type="checkbox"/> Dull	<input type="checkbox"/> Easy to use	<input type="checkbox"/> Effective
<input type="checkbox"/> Efficient	<input type="checkbox"/> Effortless	<input type="checkbox"/> Empowering	<input type="checkbox"/> Energetic
<input type="checkbox"/> Engaging	<input type="checkbox"/> Entertaining	<input type="checkbox"/> Enthusiastic	<input type="checkbox"/> Essential
<input type="checkbox"/> Exceptional	<input type="checkbox"/> Exciting	<input type="checkbox"/> Expected	<input type="checkbox"/> Familiar
<input type="checkbox"/> Fast	<input type="checkbox"/> Flexible	<input type="checkbox"/> Fragile	<input type="checkbox"/> Fresh
<input type="checkbox"/> Friendly	<input type="checkbox"/> Frustrating	<input type="checkbox"/> Fun	<input type="checkbox"/> Gets in the way
<input type="checkbox"/> Hard to Use	<input type="checkbox"/> Helpful	<input type="checkbox"/> High quality	<input type="checkbox"/> Impersonal
<input type="checkbox"/> Impressive	<input type="checkbox"/> Incomprehensible	<input type="checkbox"/> Inconsistent	<input type="checkbox"/> Ineffective
<input type="checkbox"/> Innovative	<input type="checkbox"/> Inspiring	<input type="checkbox"/> Integrated	<input type="checkbox"/> Intimidating
<input type="checkbox"/> Intuitive	<input type="checkbox"/> Inviting	<input type="checkbox"/> Irrelevant	<input type="checkbox"/> Low Maintenance
<input type="checkbox"/> Meaningful	<input type="checkbox"/> Motivating	<input type="checkbox"/> Not Secure	<input type="checkbox"/> Not Valuable
<input type="checkbox"/> Novel	<input type="checkbox"/> Old	<input type="checkbox"/> Optimistic	<input type="checkbox"/> Ordinary
<input type="checkbox"/> Organized	<input type="checkbox"/> Overbearing	<input type="checkbox"/> Overwhelming	<input type="checkbox"/> Patronizing
<input type="checkbox"/> Personal	<input type="checkbox"/> Poor quality	<input type="checkbox"/> Powerful	<input type="checkbox"/> Predictable
<input type="checkbox"/> Professional	<input type="checkbox"/> Relevant	<input type="checkbox"/> Reliable	<input type="checkbox"/> Responsive
<input type="checkbox"/> Rigid	<input type="checkbox"/> Satisfying	<input type="checkbox"/> Secure	<input type="checkbox"/> Simplistic

Legend

Positive		Neutral		Negative	
-----------------	--	----------------	--	-----------------	--

Note: The participants were given the table of cards without indicating, which ones we consider positive, negative, or neutral, in order not to influence their opinions. The classification is shown in this appendix as an indication of how we analyzed the results. These cards were created by Benedek & Miner (2002).