Alexey Tschudnowsky

End-User Development of Web-based Decision Support Systems

Doctoral Dissertations in Web Engineering and Web Science
Volume 4

Prof. Dr.-Ing. Martin Gaedke (Series Editor)

Alexey Tschudnowsky

End-User Development of Web-based Decision Support Systems



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Universitätsverlag Chemnitz
2017

Impressum

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über http://dnb.d-nb.de abrufbar.

Titelgrafik: Alexey Tschudnowsky
Satz/Layout: Alexey Tschudnowsky

Technische Universität Chemnitz/Universitätsbibliothek
Universitätsverlag Chemnitz
09107 Chemnitz
http://www.tu-chemnitz.de/ub/univerlag

readbox unipress
in der readbox publishing GmbH
Am Hawerkamp 31
48155 Münster
http://unipress.readbox.net

ISSN 2199-5354 print - ISSN 2199-5362 online

ISBN 978-3-96100-014-2

http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-21982



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik
Distributed and Self-Organizing Systems Group

End-User Development of Web-based Decision Support Systems

Dissertation

submitted in fulfillment of the
requirements for the degree of

Doktoringenieur (Dr.-Ing.)

by

Dipl.-Inf. Alexey Tschudnowsky

*December 11, 1985 in Dnepropetrowsk, Ukraine

February 3, 2017

Dissertation Committee:

Prof. Dr.-Ing. Martin Gaedke
Prof. Florian Daniel, PhD

Dipl.-Inf. Alexey Tschudnowsky
End-User Development of Web-based Decision Support Systems
Dissertation Committee:
Prof. Dr.-Ing. Martin Gaedke (Technische Universität Chemnitz, Germany),
Prof. Florian Daniel, PhD (Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy)
Submitted on September 2, 2016
Defended on February 3, 2017

Technische Universität Chemnitz
Fakultät für Informatik
Distributed and Self-Organizing Systems Group
Straße der Nationen 62
09111 Chemnitz

Abstract

Recent innovations in the information technology and computing devices magnified the volume of available information. Today's decision makers face the challenge of analyzing ever more data in shorter timeframes. Demand for technology that can efficiently assist systematic data analysis is constantly growing. Development of dedicated information systems is, however, difficult both from organizational and technological point of view. First, traditional software production is a complex and time-consuming process that can not be performed under time-pressure. Second, changing business conditions and evolving stakeholder needs require solutions that can be efficiently tailored over time. Finally, costs of custom software development are high, so that not all use cases and scenarios can be covered sufficiently.

This thesis proposes a holistic approach to address the challenges above and to enable efficient development of decision support software. The main idea is to empower end users, i.e., decision makers, in constructing their own case-specific solutions. The proposed approach called Web-Composition for End-User Development consists of a systematic process for development and evolution of decision support systems, assistance mechanisms to address lack of programming skills by decision makers and evolution facilities to enable cost- and time-efficient extensibility of user-produced solutions. The thesis describes implementation of the devised principles and ideas in the context of several open-source projects and application scenarios. Applicability and usability of the concepts are demonstrated in user studies with respective target groups. Based on the outcome analysis the thesis concludes that end users can and should actively participate in construction of decision support software.

In loving memory of my father Roman Chudnovskyy
1951-2014

Acknowledgments

This doctoral dissertation is the result of a six years research and would not have been possible without the support and advice of many people. In the following I would like to express my sincere thank to all of them.

First of all, i thank my supervisor Prof. Dr.-Ing. Martin Gaedke for his caring guidance and continuous support throughout the thesis. In a friendly but demanding atmosphere Prof. Gaedke helped me to develop skills required to conduct scientific research. He encouraged and guided me, his knowledgeable feedback and recommendations contributed to the success of numerous research activities. I am also grateful to my second supervisor Prof. Florian Daniel, PhD for his extensive and instructive feedback on the thesis. Publications and research of Prof. Daniel inspired parts of my work and provided me with insights on high quality scientific work.

I sincerely thank my wife Olga, my parents Nina and Roman, my brother Pawel for their steadfast faith in my success. This dissertation would be impossible without their encouragement, patience and support throughout all these years.

I thank all former colleagues and friends at VSR – Stefan Wild, Ralph Sontag, Dr.-Ing. Jörg Anders, Michael Krug, Sebastian Heil, Fabian Wiedemann, Hendrik Gebhardt, Frank Weinhold and Bahareh Zarei – for the friendly atmosphere, joint publications, amusing discussions and continuous knowledge exchange during my PhD study. I am also

grateful to Dr.-Ing. Matthias Heinrich and Dr.-Ing. Stefan Pietschmann for the recommendations and inspirations they provided me with during our joint research activities.

Finally, I would like to thank to the many students whom I worked with in these six years – Sebastian Müller, Christian Fischer, Philipp Schmiedel, Michael Hertel, Masha Didkovska, Martin Sommer and many others. Their dedicated work yielded several joint publications and significantly contributed to the results of this thesis.

Publications

The thesis builds upon materials of the following publications[1].

1. Chudnovskyy, Olexiy and Martin Gaedke (2010). "Development of Web 2.0 Applications using WebComposition / Data Grid Service". In: *The Second International Conferences on Advanced Service Computing (Service Computation 2010)*. Ed. by Ali Beklen, Jorge Ejarque, and Wolfgang Gentzsch. **Best Paper Award**. Lisbon, Portugal: IARIA, pp. 55–61.

2. Chudnovskyy, Olexiy, Sebastian Brandt, and Martin Gaedke (2011). "Integrating Human-services Using WebComposition/UIX". In: *Proceedings of the Workshop on Posters and Demos Track*. PDT '11. Lisbon, Portugal: ACM, 21:1–21:2.

3. Chudnovskyy, Olexiy, Hendrik Gebhardt, Frank Weinhold, and Martin Gaedke (2011). "Business Process Integration using Telco Mashups". In: *Procedia Computer Science* 5. The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011), pp. 677–680.

[1] In November 2013 the name of the author changed from "Chudnovskyy, Olexiy" to "Tschudnowsky, Alexey"

4. Chudnovskyy, Olexiy, Frank Weinhold, Hendrik Gebhardt, and Martin Gaedke (2011). “Integration of Telco Services into Enterprise Mashup Applications.” In: *ICWE Workshops*. Ed. by Andreas Harth and Nora Koch. Vol. 7059. Lecture Notes in Computer Science. Springer, pp. 37–48.

5. Weinhold, Frank, Olexiy Chudnovskyy, Hendrik Gebhardt, and Martin Gaedke (2011). “Geschäftsprozessintegration auf Basis von Telco-Mashups”. In: *INFORMATIK 2011*. Ed. by Pepper Heiß and Schneider Schlingloff. Berlin, Germany: Gesellschaft für Informatik e.V. (GI), p. 376.

6. Chudnovskyy, Olexiy and Martin Gaedke (2012). “End-User-Development and Evolution of Web Applications: The WebComposition EUD Approach”. In: *Current Trends in Web Engineering*. Ed. by Michael Grossniklaus and Manuel Wimmer. Vol. 7703. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 221–226.

7. Chudnovskyy, Olexiy, Sebastian Müller, and Martin Gaedke (2012). “Extending Web Standards-Based Widgets towards Inter-Widget Communication”. In: *Current Trends in Web Engineering*. Ed. by Michael Grossniklaus and Manuel Wimmer. Vol. 7703. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 93–96.

8. Chudnovskyy, Olexiy, Tobias Nestler, et al. (2012). “End-User-Oriented Telco Mashups: The OMELETTE Approach”. In: *Proceedings of the 21st International Conference Companion on World Wide Web*. WWW ’12 Companion. New York: ACM, pp. 235–238.

9. Chudnovskyy, Olexiy, Stefan Wild, Hendrik Gebhardt, and Martin Gaedke (2012). “Data Portability Using WebComposition/Data Grid Service”. In: *International Journal on Advances in Internet Technology* 4.3 & 4, pp. 123–132.

10. Chudnovskyy, Olexiy, Christian Fischer, Martin Gaedke, and Stefan Pietschmann (2013). “Inter-Widget Communication by Demonstration in User Interface Mashups”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 502–505.

11. Chudnovskyy, Olexiy, Stefan Pietschmann, Matthias Niederhausen, Vadim Chepegin, et al. (2013). “Awareness and Control for Inter-Widget Communication: Challenges and Solutions”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 114–122.

12. Roy Chowdhury, Soudip, Olexiy Chudnovskyy, Matthias Niederhausen, Stefan Pietschmann, et al. (2013). “Complementary Assistance Mechanisms for End User Mashup Composition”. In: *Proceedings of the 22nd International Conference on World Wide Web Companion*. WWW ’13 Companion. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, pp. 269–272.

13. Wild, Stefan, Olexiy Chudnovskyy, Sebastian Heil, and Martin Gaedke (2013a). “Customized Views on Profiles in WebID-Based Distributed Social Networks”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Heidelberg: Springer, pp. 498–501.

14. Wild, Stefan, Olexiy Chudnovskyy, Sebastian Heil, and Martin Gaedke (2013b). “Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering”. In: *Current Trends in Web Engineering*. Ed. by Quan Z. Sheng and Jesper Kjeldskov. Vol. 8295. Lecture Notes in Computer Science. Springer, pp. 269–280.

15. Tschudnowsky, Alexey, Michael Hertel, Fabian Wiedemann, and Martin Gaedke (2014). “Towards Real-time Collaboration in User Interface Mashups”. In: *ICE-B 2014 - Proceedings of the 11th In-*

ternational Conference on e-Business. Vienna, Austria, pp. 193–200.

16. Tschudnowsky, Alexey, Stefan Pietschmann, Matthias Niederhausen, and Martin Gaedke (2014). "Towards Awareness and Control in Choreographed User Interface Mashups". In: *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*. WWW Companion '14. Seoul, Korea: International World Wide Web Conferences Steering Committee, pp. 389–390.

17. Tschudnowsky, Alexey, Stefan Pietschmann, Matthias Niederhausen, Michael Hertel, and Martin Gaedke (2014). "From Choreographed to Hybrid User Interface Mashups: A Generic Transformation Approach". In: *Web Engineering*. Ed. by Sven Casteleyn, Gustavo Rossi, and Marco Winckler. Vol. 8541. Lecture Notes in Computer Science. Springer International Publishing, pp. 145–162.

18. Hertel, Michael, Alexey Tschudnowsky, and Martin Gaedke (2015). "Conflict Resolution in Collaborative User Interface Mashups". In: *Engineering the Web in the Big Data Era*. Ed. by Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe. Vol. 9114. Lecture Notes in Computer Science. Springer International Publishing, pp. 659–662.

19. Tschudnowsky, Alexey and Martin Gaedke (2015). "Loop Discovery in Publish-Subscribe-Based User Interface Mashups". In: *Engineering the Web in the Big Data Era*. Ed. by Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe. Vol. 9114. Lecture Notes in Computer Science. Springer International Publishing, pp. 683–686.

20. Wild, Stefan, Fabian Wiedemann, Sebastian Heil, Alexey Tschudnowsky, and Martin Gaedke (2015). "ProProtect3: An Approach for Protecting User Profile Data from Disclosure, Tampering, and Improper Use in the Context of WebID". In: *Transactions on Large-*

Scale Data- and Knowledge-Centered Systems. Lecture Notes in Computer Science 8990: *Special Issue on Big Data and Open Data XIX*. Ed. by Abdelkader Hameurlain, Josef Küng, Roland Wagner, Devis Bianchini, et al., pp. 87–127.

Contents

1 Introduction **1**
- 1.1 Decision Making under Time Pressure 1
- 1.2 Motivation 2
- 1.3 Problem Statement 3
- 1.4 Research Objectives 4
- 1.5 Research Contributions 4
- 1.6 Research Method 5
- 1.7 Scope of the Thesis 7
- 1.8 Structure of the Thesis 8
- 1.9 Summary 8

2 Requirements Analysis **9**
- 2.1 Scenarios 9
  - 2.1.1 Emergency Response 9
  - 2.1.2 Ad-hoc Reporting 11
  - 2.1.3 Accommodation Search 12
- 2.2 Stakeholder 14
- 2.3 Requirements 15
  - 2.3.1 Development Process 15
  - 2.3.2 Tool Assistance 17
- 2.4 Summary 20

3 State of The Art **23**
- 3.1 Development Processes 23
  - 3.1.1 End-User Development 24

3.1.2 Component-Based Development 27
3.1.3 Model-Driven Development 35
3.2 Tool Assistance 40
3.2.1 Web Content Management Systems 41
3.2.2 Dashboards and Ad-Hoc Reporting Tools 44
3.2.3 Composition Tools 49
3.3 Discussion 59
3.4 Summary 61

4 WebComposition/EUD Approach 63
4.1 Overview 63
4.2 Principles 65
4.3 Formalisms 67
4.3.1 Component Model 67
4.3.2 Composition Model 71
4.4 Process Model 76
4.5 Methods 81
4.6 Tools 82
4.6.1 Composition Platform 83
4.6.2 Development Assistance 83
4.6.3 Evolution Assistance 84
4.7 Summary 86

5 Composition Platform 89
5.1 Research Questions 89
5.2 Requirements 90
5.3 Conceptual Architecture 91
5.4 Implementation 93
5.4.1 WebComposition/EUD Components 93
5.4.2 WebComposition/EUD Composition 98
5.4.3 Run-Time Environment 101
5.4.4 Live Composition Editor 104
5.5 Evaluation 110
5.5.1 Awareness and Control Facilities 111
5.5.2 Transformation Editor 115
5.6 Summary 117

6 Development Assistance **119**
6.1 Research Questions 119
6.2 Automatic Discovery and Composition Engine 120
6.2.1 Motivation Scenario 121
6.2.2 Requirements 122
6.2.3 Automatic Discovery and Composition 122
6.2.4 Related Work 128
6.2.5 Evaluation 129
6.3 Loop Detection Facilities 133
6.3.1 Motivation Scenario 133
6.3.2 Requirements 134
6.3.3 Loop Discovery 135
6.3.4 Related Work 141
6.3.5 Evaluation 142
6.4 Double Input Detector 143
6.4.1 Motivation Scenario 144
6.4.2 Requirements 144
6.4.3 Automation of User Input 145
6.4.4 Related Work 151
6.4.5 Evaluation 152
6.5 Summary 155

7 Evolution Assistance **157**
7.1 Research Questions 158
7.2 WebComposition/EUD Component Converter 158
7.2.1 Motivation Scenario 159
7.2.2 Requirements 160
7.2.3 Conversion Process 160
7.2.4 Related Work 166
7.2.5 Evaluation 167
7.3 WebComposition/EUD ICCI Extender 169
7.3.1 Motivation Scenario 170
7.3.2 Requirements 170
7.3.3 Semi-automatic ICCI Extension 171
7.3.4 Related Work 177
7.3.5 Evaluation 178
7.4 WebComposition/EUD Artifact Library 184

7.4.1 Motivation Scenario . . . 185
7.4.2 Requirements . . . 185
7.4.3 Artifact Access and Management . . . 186
7.4.4 Related Work . . . 196
7.4.5 Evaluation . . . 197
7.5 Summary . . . 199

8 Overall Evaluation 201
8.1 Requirements Evaluation . . . 201
8.1.1 Development Process . . . 201
8.1.2 Development Toolkit . . . 203
8.2 Application Scenarios . . . 207
8.2.1 Public Information Screen . . . 207
8.2.2 Collaborative Decision Making . . . 208
8.2.3 Telecommunication Dashboards . . . 210
8.3 Summary . . . 212

9 Conclusions and Outlook 213
9.1 Summary of the Thesis . . . 213
9.2 Lessons Learned . . . 215
9.3 Summary of Contributions . . . 216
9.4 Ongoing and Future Work . . . 217
9.4.1 Toolkit Improvements . . . 218
9.4.2 Open Questions . . . 220

A Schemes 249
A.1 XSD schema of the proposed W3C configuration document extension . . . 249
A.2 XSD schema of the proposed OMDL extension . . . 251

B Evaluation Materials 255
B.1 Awareness and Control Facilities . . . 255
B.1.1 Questionnaire . . . 255
B.1.2 Results . . . 258
B.2 Transformation Editor . . . 259
B.2.1 Questionnaire . . . 259
B.2.2 Results . . . 261
B.3 Automatic Discovery and Composition Engine . . . 262

B.3.1 Questionnaire 262
B.3.2 Results 264
B.4 WebComposition/EUD ICCI Extender 265
B.4.1 Questionnaire 265
B.4.2 Results 268
B.5 Performance Evaluation of the WebComposition/EUD Artifact Library 269

Figures **271**

Tables **275**

Listings **277**

Abbreviations

ADCE Automatic Discovery and Composition Engine

API Application Programming Interface

BI Business Intelligence

BPEL Business Process Execution Language

CASE Computer-aided Software Engineering

CDB Component-Based Development

CBSE Component-Based Software Engineering

CBWE Component-Based Web Engineering

COM Component Object Model

CRM Customer Relationship Management

CRUD Create, Read, Update, Delete

CRUDS Create, Read, Update, Delete, Search

CSS Cascading Style Sheets

DGS Data Grid Service

DID Double Input Detector

DOM Document Object Model

DSL Domain Specific Language

DSS Decision Support System

EIS Executive Information System

EJB Enterprise Java Beans

ERP Enterprise Resource Planning

EUD End-User Development

EUSE End-User Software Engineering

GUI Graphical User Interface

HCI Human-Computer Interaction

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ICC Inter-Component Communication

ICCI Inter-Component Communication Interface

IRI Internationalized Resource Identifier

IT Information Technology

IWC Inter-Widget Communication

JSON Javascript Object Notation

KPI Key Performance Indicator

LDF Loop Detection Facilities

MCDM Multiple Criteria Decision Making

MDD Model-Driven Development

MIS Management Information Systems

MVC Model-View-Controller

NLP Natural Language Programming

OLAP Online Analytical Processing

OMELETTE Open Mashup Enterprise service platform for LinkEd data in The TElco domain

OMDL Open Mashup Description Language

PBD Programming-by-Demonstration

PIM Platform Independent Model

PSM Platform Specific Model

REST Representational State Transfer

RDF Resource Description Framework

ROA Resource-Oriented Architecture

ROI Return of Investment

RSS Really Simple Syndication

SHDM Semantic Hypermedia Design Method

SOA Service-Oriented Architecture

SPARQL SPARQL Protocol And RDF Query Language

UI User Interface

UML Uniform Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

UWA Universal Web App

UWE UML-based Web Engineering

V&V Verification and Validation

W3C World Wide Web Consortium

WAC Web Access Control

WADL Web Application Description Language

WebComposition/EUD-AL WebComposition/EUD Artifact Library

WebComposition/EUD-CC WebComposition/EUD Component Converter

WebComposition/EUD-IE WebComposition/EUD ICCI Extender

WebComposition/EUD WebComposition for End-User Development

WCMS Web Content Management Systems

WebML Web Modeling Language

WebRE Web Requirements Engineering

WebSA Web Software Architecture

WPF Windows Presentation Foundation

WSDL Web Service Description Language

WSA Web Services Architecture

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

XSD XML Schema Definition

XSLT XSL Transformations

1 Introduction

Today's competitive economy requires decision makers to analyze higher volume of data in ever shorter periods of time. In 2005 data needed for decision making came from 6-10 sources (Hurwitz et al., 2005). Since the spread of Social Media and its extensive use for decision making the number of relevant data sources increased significantly (Zeng et al., 2010). At the same time decision makers are given ever less time for information analysis. Shortened product delivery times and accelerated internal processes constantly put employees under high time pressure (Trinkfass, 1997). Informed decision making becomes more challenging and less systematic (Isenberg, 1984). To avoid negative effects on performance of decision makers new methods and assisting technologies are required (Forrester Research, 2015).

1.1 Decision Making under Time Pressure

Decision making can be defined as a "study of identifying and choosing alternatives based on the goals, objectives, desires, values, and preferences of decision makers" (Harris, 2012). Decision making usually results in selection of course of action among several alternative options or scenarios. It is a reasoning or emotional process, rational or irrational, can be based on explicit or tacit assumptions. In contrast to unconscious

decision making, which is often acceptable in everyday life (Nightingale, 2007), science, industry and politics demand systematic, transparent and justifiable methods (Baker et al., 2001).

As shown by studies systematic decision making is difficult to apply under time-pressure (J. R. Busemeyer and A. Diederich, 2002; Isenberg, 1984; A. Diederich and J. R. Busemeyer, 2003). **Time-pressuring situations** are characterized by strict deadlines, limited resources and information overload that distort otherwise systematic and objective decision making (Boundless, 2013). Available information and alternative options are analyzed superficially, which often results in lower quality of decisions (Maule and Andrade, 1997). Under time-pressure individuals tend to rely on heuristics and rules of thumb instead of on systematic and thorough analysis methods (Gigerenzer and Todd, 2000; Boussemart et al., 2009).

1.2 Motivation

To enable systematic decision making under time pressure effective and efficient information systems are required (Hwang, 1994). Software industry produced many solutions to address different scenarios and activities (Hedgebeth, 2007). So called **Decision Support Systems (DSSs)** provide "communication technologies, data, documents, knowledge and/or models to identify and solve problems, complete decision process tasks, and make decisions" (D. J. Power, 2002). Business Intelligence (BI), data warehousing, Online Analytical Processing (OLAP) software, expert systems and Management Information Systems (MIS) (also known as Dashboards) are widely used systems to improve quality of decisions and to optimize operational efficiency (Kielstra et al., 2007; Brynjolfsson et al., 2011).

The large number of different systems highlights demand for technological assistance in decision making. However, time-pressuring situations demand specific solutions tailored towards unique requirements (Hwang, 1994). Although existing systems can be customized

and extended towards new use cases, the process of customization itself is a time-consuming and error-prone activity (Kumar, 2013). Usually it is done by professional software developers that have corresponding skills and education. Due to delays and produced costs delegation of development activities is often not feasible. *Ability of systems to be customized or developed directly by decision makers becomes crucial and often the only option to obtain an appropriate software in the emerged situation* (D. J. Power, 2002).

1.3 Problem Statement

The central problem addressed by this thesis is the *high amount of work, time and programming skills required for development of case-specific DSSs*. The latter are software solutions that provide data and functionalities required for situational decision making under time-pressure. The main effect of the problem is low performance of decision makers under time-pressure due to absence of adequate software assistance. The central problem is mainly caused by the following three subproblems:

Delays and Expenses Due to Delegation of Development Activities Software development requires programming skills and education that decision makers usually do not have (Repenning and Ioannidou, 2006). As a result, actual product development is usually delegated from decision makers (end users) to software vendors and, thus, becomes costly. The delegation requires additional steps such as domain exploration, requirements analysis, software validation etc. that delay the development process. Potential communication problems between customer and team can make it even longer and more expensive (Gachet and Haettenschwiler, 2006).

Lack of Methodological and Technological Assistance Current methods and tools for software development address time-pressuring situations only insufficiently (Hwang, 1994). Under tight time constraints it is challenging to perform all development activities

to their full extent. The latter, however, can lead to poor quality of the end product. Little research has been performed on systematic software development under time-pressure (Austin, 2001).

Poor Maintainability of Software Developed Under Time-Pressure Developing a software that can be continuously tailored towards new use cases is a difficult task. Design of a maintainable and evolution-aware architecture becomes even more difficult if performed under time-pressure (Lehman and Ramil, 2003). Improperly or inefficiently designed software causes high expenses during its whole life-cycle (Madhavji et al., 2006).

1.4 Research Objectives

The goal of this thesis is to *enable time- and cost-efficient development of case-specific DSSs under time-pressure*. In achieving this goal the thesis addresses three main objectives:

Objective1. Enable end users to develop case-specific DSSs with minimal to no involvement of professional software developers.

Objective2. Provide methodological and technological assistance to perform software development under time pressure.

Objective3. Enable time- and cost-efficient evolution of software solutions that were developed under time-pressure.

1.5 Research Contributions

The main contribution of this thesis is *a methodology for time- and cost-efficient development of case-specific DSSs* that can be applied by non-programmers under time-pressure. The thesis results in a number of developments to address the objectives stated above:

Enabled development of DSSs by end users Based on an analysis of existing state-of-the-art technologies the thesis defines a UI-centric composition-based approach for development of DSSs directly by decision makers (Chapter 4). The approach consists of quality-ensuring principles, formalisms, methods, tools and a process model. Furthermore, it provides a platform (cf. Chapter 5) that implements the proposed concepts and enables their evaluation in user studies.

Accelerated development process The thesis develops three end-user assistance mechanisms to speed-up construction of DSSs in time-pressuring situations (cf. Chapter 6). First, a dialog-based expert system is devised, whose goal is to automate discovery and composition of partial solutions based on high-level business goals and, thus, to accelerate the development process. Second, algorithms for detection of faulty configurations are developed that improve system reliability while construction under time-pressure. Finally, the Double Input Detector facility speeds up user inputs by applying automatically learned interaction patterns.

Costs-optimized evolution of end-user-developed software The thesis explores methods for efficient maintenance and evolution of end-user-produced DSSs (cf. Chapter 7). An algorithm for automatic inclusion of new data sources and functionalities based on proprietary software components is devised. Assistance mechanisms to enrich existing functionalities with more flexible integration capabilities are developed. Finally, a reusable repository for efficient management, sharing and discovery of software artifacts is provided.

1.6 Research Method

The research has been conducted based on the following approach:

- A literature study has been performed with the goal of identifying obstacles and their causes for software development under time-pressure.

- Thesis objectives have been formulated with focus on empowering decision makers to quickly create DSSs with minimal to no involvement of professional developers.

- Several scenarios highlighting demand for assisted development of DSSs by decision makers have been identified and stakeholder analysis has been performed. The two enabled elicitation of requirements on the appropriate end-user involvement methodology and related technological assistance.

- Current state of the art in Software- and Web Engineering as well as End-User Development (EUD) has been analyzed and fulfillment of stated requirements by existing technology evaluated. Non-addressed challenges to be considered further in the thesis have been identified.

- Based on the stated requirements and identified deficiencies in the state of the art a theoretical foundation for end-user-oriented DSS development platform and accompanying assistance mechanisms has been designed.

- Prototypes of the platform and of assistance mechanisms were implemented. Their purpose was 1) to proof validity of the design and 2) to enable collection of data required for overall approach evaluation.

- Achievement of the stated objectives and quality / efficiency of the developed prototypes were evaluated within user studies conducted in form of laboratory experiments.

- Research results were disseminated in form of scientific publications, presentations in workshops and conferences as well as contributions to communities of utilized open source projects.

1.7 Scope of the Thesis

This thesis is mainly settled on the area of Web Engineering with several cross-cutting concerns from Human-Computer Interaction (HCI) and EUD. The following research questions were not considered in its scope:

- The thesis doesn't consider systems that support decision making by manipulating simulation models, performing document management, enabling communication and collaboration or providing expert capabilities. It rather focuses on solutions that enable efficient access to data and functionalities required for case-specific decision making.

- Usability and User Interface (UI) design are important aspects for acceptance and success of software. This thesis partially addresses this topic by developing mechanisms for aggregation and manipulation of components by end users. However, the responsibility of creating appealing and responsive UI design, corporate or uniform Look-and-Feel lies on component developers.

- Development of Web-based DSSs that make use of several connected devices, enables new usage scenarios and provides novel user experience. Although this thesis focuses on one-screen DSSs, its findings have been partially reused and extended towards multi-screen scenarios in a German national research project Chrooma+ (Krug et al., 2013).

- Automatic reconfiguration of user-developed solutions depending on new context or changed requirements has been addressed by other research projects such as CRUISE (Pietschmann, Radeck, et al., 2011). This thesis provides extension points to implement adaptivity based on components and composition models.

1.8 Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 introduces motivation scenarios and key stakeholders, affected by the central problem. Based on the scenarios and characteristics of the stakeholders, requirements on an appropriate DSS development method are elicited. Chapter 3 reviews state of the art of existing technologies that have a goal to enable time- and cost-efficient development of software solutions. Chapter 4 presents a conceptual view on the proposed DSS development and evolution method with focus on strong end-user involvement. The subproblems described in Section 1.3 are addressed in the subsequent chapters. Chapter 5 introduces the composition platform that enables non-technical users to build and configure DSS out of a library of reusable components. Chapter 6 presents assistance mechanisms to address the time-pressure constraint and to speed-up DSS construction process. Afterwards, in Chapter 7 evolution assistance mechanisms are described. Overall approach evaluation can be found in Section Chapter 8. The thesis and its contributions are summarized in Section Chapter 9, which also identifies non-addressed challenges and points out further research directions.

1.9 Summary

This chapter introduced the context of the thesis and motivated research on systematic DSS development under time pressure. Challenges in enabling non-technical end users to obtain case-specific information system under time-pressure have been identified. The chapter presented objectives and contributions of the thesis. Research method and scope were given. The following chapter explores the problem domain by identifying its key stakeholders and eliciting requirements on a quality-ensuring development approach.

2 Requirements Analysis

This chapter describes requirements on an appropriate solution of the central problem. First, guiding scenarios from three different domains are introduced. Then, involved stakeholders are identified. Finally, requirements on the development process and assisting technologies are elicited.

2.1 Scenarios

The following three scenarios highlight importance of ability to quickly develop a case-specific DSS in time-pressuring situations.

2.1.1 Emergency Response

One of the responsibilities of emergency response agencies is to coordinate rescue activities in case of hurricanes, fire, flood, earthquakes or nature catastrophes. Fast and professional reaction in response to emergency situations is crucial to minimize human and material losses. To initiate appropriate rescue activities a profound analysis of the concrete emergency case should be conducted. Scale of damages, course

of events but also available rescue resources should be analyzed. The corresponding data has to be collected quickly and rescue activities initiated immediately (Bakonyi et al., 2008).

A concrete example of an emergency situation is the flood of Dresden in Saxony, Germany in 2002. For coordination teams it was crucial to quickly get an overview of water levels in different geographical regions of the affected area, identify objects at risk, resident vulnerability etc. For this purpose plenty of information sources including social media reports had to be analyzed. Reliable and efficient communication with police, fire and ambulance services was required to coordinate rescue activities.



Figure 2.1.: FireView Dashboard[1]: A Software Solution for Fire and Emergency Response Agencies

Various systems exist for expected and well-understood scenarios such as fire response (cf. Figure 2.1). However, as indicated by (Bakonyi et al., 2008), end-user requirements on flood management systems vary "according to their responsibilities and technical capacity". Members of coordination team require customized tools that address their current information and communication needs. While the flood situation changes, team members might need to access new and unforeseen informations

[1] `http://www.theomegagroup.com/fire/omega_dashboard_fire.html`, Retrieved: 5.7.2015

sources such as user-generated content or streams from specific web cameras. The flood management system in use might lack the required functionalities. Extending it towards new requirements is, however, a complex and time-consuming process that is usually performed by professional programmers. Team members do not have required skills and time to appropriately extend the software during an emergency situation. Lack of a possibility to perform fast customizations and extensions towards unexpected situations results in decreased performance of the team, longer response times and less efficient rescue process.

2.1.2 Ad-hoc Reporting

Ad-hoc reporting is a BI process, in that data required for answering business-related questions is collected and visualized on demand, i.e., depending on concrete situation or requirements. In contrast to software solutions or reports configured by professional software developers, ad-hoc reporting is usually performed by users without technical skills – managers, executives, administrative stuff etc. The goal of ad-hoc reporting is usually twofold: First, business users should be empowered to collect and analyze data for a wide range of business questions and second, company IT should be unburdened from tedious and constantly changing reporting tasks.

Consider a management meeting in a company, whose business is production and selling of winter sport equipment across Europe. The goal of the meeting is to evaluate Key Performance Indicators (KPIs) of the most recent quarter and to adjust the marketing strategy correspondingly. Many management information systems enable collection and presentation of data based on templates and data sources as they were foreseen by software providers. Figure 2.2 shows an example Web application for aggregation and evaluation of company sales data.

However, questions raised in the meeting can be very different and unexpected. One might be interested in feedback about newly intro-

[2] `http://www.jinfonet.com/solutions/dashboard`, Retrieved: 4.7.2015

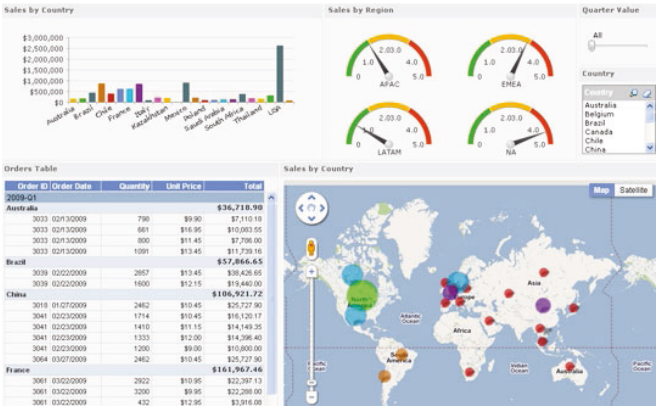


Figure 2.2.: JReport Dashboards[2]: A Web Application for Visualization of KPIs

duced products in social media or in press. One might want to compare product characteristics with those of competitors. Finally, one might need to get in touch with people being currently not in the meeting and to discuss the retrieved data with them.

Extending an existing system towards new use cases is a time-consuming task that requires time and expertise in programming. Traditional software development methods are too complex to be applied during the management meeting. Furthermore, it can be costly and not feasible to change the software for every new question raised. Without adequate software support, however, quality of decisions made and overall efficiency of the meeting significantly decrease (Sauter, 2014).

2.1.3 Accommodation Search

A total of 9,39 million Germans changed their place of living in 2013[3]. Choice of a new flat or a new house is usually driven by a number of requirements on the accommodation itself and on its neighborhood. To

[3]Source: ummelden.de, `http://www.ummelden.de/umziehen-in-deutschland-daten-fakten-2014.html`, Retrieved: 4.7.2015

find the “best match” one usually collects available data first, compares and analyzes best-suited candidates, shares the findings with friends or colleagues and gets in touch with the landlord. Available software assistance significantly defines how time-consuming the evaluation process will be.

For example, consider a situation, in which Alex, a Web designer from Dresden, finds a new job position in Berlin and moves to the new city. A real estate agent offered him several alternatives and Alex has to decide fast, as the demand for flats in Berlin is strong. Alex opens a real state search engine and compares the different offers (cf. Figure 2.3). It is important for him to live in a green area with little crime, to have sufficient shopping possibilities and to have a good public transport connection to the new office.

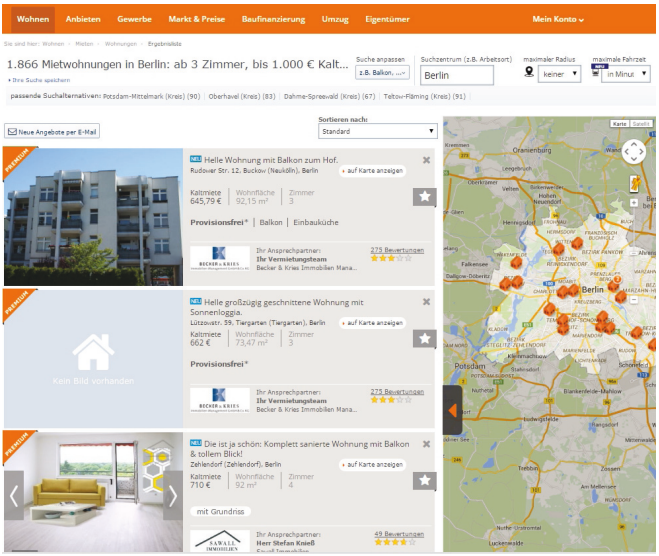


Figure 2.3.: Immobilienscout24[4]: A Web Application for Real Estate Management

[4] `http://www.immobilienscout24.de`, Retrieved: 4.7.2015

Because the search engine doesn't provide all the information he requires, Alex visits additionally a picture gallery of Berlin's streets, the homepage of public transport agency and Web pages with statistical data about pollution and crime in the city. The Web applications require him to input location data several times, aggregate results manually and switch back and forth between different contexts. If Alex wants to discuss his findings with friends or family, he has to start yet another application such as voice chat or email client and copy-paste interesting information into it.

The example shows that manual aggregation and presentation of data from heterogeneous sources is time-consuming and error-prone. *End users demand software solutions that would help to answer complex search queries with as little manual actions as possible* (Ceri, 2009). However, in many cases users do not have access to dedicated solutions (or such solutions do not exist yet). To address lack of specific solutions, it is desirable to enable users to quickly develop required solutions on their own.

2.2 Stakeholder

Based on the above scenarios two main stakeholder groups can be identified that are affected by the central problem:

Decision Makers (End Users) This group comprises department managers, knowledge workers, administrative stuff but also individuals from a variety of domains that apply structured and systematic decision making processes to solve situational problems, perform strategic planning or coordinate group activities (Hodgkinson and Starbuck, 2008). Decision makers are domain experts from all age groups that possess knowledge, experience and skills in the area of their specialization, e.g., medicine, project management, accounting, emergency response, commerce etc. They are familiar with operation of office software, Web browsers and PC / mobile operating systems. However, being non-programmers, decision

makers usually lack skills for algorithmic thinking, understanding of source code or technical software models, dealing with data structures and representations.

Software Providers This group comprises software companies, individual developers and IT support staff, who are in charge of provisioning and maintenance of organizational software infrastructure as well as providing technical support. Usually software providers have skills and education in programming and administration, apply systematic processes and tools for software development (Tockey, 1999). Software providers often lack domain knowledge required for deep understanding of customer problems. Communication and documentation is therefore coined by precise models and abstractions that aim at better understanding of problem domain and minimizing ambiguity (often caused by natural language).

2.3 Requirements

To enable time-efficient development of DSS by non-programmers two groups of requirements have been identified: the first group focuses on the development process and the second one on enabling tools and assisting technologies. The requirements are elicited based on the thesis objectives, the presented scenarios and stakeholder characteristics. To assess the degree of requirement satisfaction a scale with 3 values has been introduced (fully satisfied, partially satisfied, not satisfied). Semantics of the values in context of each requirement depends on the objectives of the thesis.

2.3.1 Development Process

The introduced scenarios highlighted situations that demand situational software solutions, but do not let much time for systematic design, programming or quality preservation activities. Thus, an enabling deve-

lopment process should involve decision makers as much as possible (to preserve domain-expertise and avoid time-consuming communication), be concise (to address the time-constraint) and build upon existing tools and building blocks (to increase quality and maintainability of resulting solutions). These considerations lead to the following requirements on the development process:

D1: End-User Involvement: Decision makers should actively participate in development and evolution of DSSs. As discussed in Section 1.2 shift of these activities towards end users would potentially result in a number of advantages both from operational but also economical point of view. The challenge is, however, to identify activities that can and should be performed by non-professionals without impacting security, reliability or performance of the whole IT infrastructure (Ye and Gerhard Fischer, 2007). It is desirable to involve end users in as much development activities as possible, so that fully-fledged software solutions fitting situational needs of users can be produced.

The satisfaction criterion of this requirement is the permanent involvement of end users into the development process that lets them develop, maintain and extend solutions on their own without any help. The requirement is partially fulfilled, if end users are strongly involved into development process but some activities (e.g., maintenance and evolution) are assigned to professionals. Finally, the requirement is considered to be not satisfied, if end users act solely as sources of requirements and all development and evolution activities are carried out by professional developers.

D2: Process Conciseness: In time-pressuring situations it is important to obtain a DSS as fast as possible and, thus, to leave more time for analysis and evaluation of decision making-related data. Traditionally, software development is a complex and long process that consists of many steps: system specification, design and implementation, verification, deployment (cf. Waterfall (Royce, 1987) or Spiral Models (Boehm, 1988)). Also agile processes, which aim at iterative and continuous delivery of software, foresee

time-consuming planning, testing and review activities (Martin, 2002). To produce software under time-pressure, it is desired to devise a process that would significantly shorten, simplify and even skip some of traditional steps without decreasing quality of the outcomes (Fitzgerald and Hartnett, 2005).

The satisfaction criterion for this requirement is a concise development process, which means that time-consuming steps are skipped or performed within separated processes before the actual development starts. The requirement is partially satisfied, if steps are present but are simplified or automated using dedicated software tools. Finally, the requirement is not satisfied, if all phases of traditional development process are present.

D3: Reuse-Orientation: Development of solutions from scratch is usually a cost-intensive and time-consuming process. On the other side, industries and organizations have already invested financial, material and human resources into existing software infrastructure and expect appropriate Return of Investment (ROI) (Lim, 1994). Despite of financial savings, reuse decreases time required to produce a product as well as its quality. The advantages of reuse are, however, difficult to achieve if it is not performed systematically (Schmidt, 1999). The process of DSS development should promote creation, management, discovery and reuse of software artifacts as much as possible.

The satisfaction criterion for this requirement is a strong focus on production and reuse of software artifacts. The requirement is partially satisfied if a process focuses on reuse only. Finally, the requirement is not satisfied if reuse of software artifacts is possible but neither promoted nor enforced in process activities.

2.3.2 Tool Assistance

In time-constrained situations as presented in scenarios above it is important to obtain an appropriate DSS quickly and with little effort

(development step). Once user needs evolve or situation changes the obtained solution should be changed or extended with new functionalities (evolution step). Because development tools are applied under time-pressure, non-functional properties such as usability, time-efficiency and fault-tolerance gain particular importance (D. J. Power, 2002). Based on these considerations, the requirements on assisting tools are defined as follows:

T1: Development Assistance: Tools and technologies should support development of data-driven DSSs and provide dedicated Web-based run-time environments. In particular, they should enable integration of multiple data sources and tailoring of the view to meet personal preferences or situational needs (Hurwitz et al., 2005). Ideally, integration logic should enable definition of any control and data flow on top of integrated elements. Tailoring of the view should be supported in terms of free placement of integrated data sources or functionalities and flexible configuration of application UI.

The requirement is fully satisfied if all mentioned functionalities are provided. It is partially satisfied if they are present, but some of the functionalities are limited in expressiveness or scope. The requirement is not satisfied, if one or more functionalities are completely missing.

T2: Evolution Assistance: Software evolution can be defined as a "process of progressive change and cyclic adaptation over time in terms of the attributes, behavioral properties and relational configuration of some material, abstract, natural or artificial entity or system" (Scacchi, 2006). Evolution in case of data-driven DSSs means in first line integration of new data sources and functionalities, which is required in case of changed requirements or new questions raised by decision makers (Orts, 2005). Interoperability and compatibility among the new and existing functionalities should be constantly preserved, which requires efficient updates to interfaces and logic of reusable artifacts. Finally, systematic

management of reusable software artifacts and produced solutions should be supported.

The requirement is fully satisfied if all mentioned functionalities are provided. It is partially satisfied, if only subset of functionalities is present. The requirement is not satisfied, if none of the above functionality is implemented.

T3: Ease of Use: All stakeholders should be able to focus on their business goals instead of spending mental, time and financial resources on operating technology (Hurwitz et al., 2005). In this context ease of use, case- and target-group orientation are considered to be key elements for acceptance of a technology and for satisfaction of its users. Both development and evolution assistance tools should be easy and comfortable to use, so that they can be applied even in time-pressuring situations. The tools should be easy to learn, intuitive in use and correspond to user aesthetic values or behavior habits. Complexity of utilized abstractions, languages and interaction patterns should meet skills and experiences of respective target user groups.

The requirement is considered to be fully satisfied, if tools make use of vocabulary and interaction principles that are familiar or domain-specific to respective target groups. It is partially satisfied, if concepts and interaction patterns are not domain-specific but can be easily learned given the IT skills of respective target group. Finally, it is not satisfied, if tools use concepts and patterns that impose too high challenges on their target groups.

T4: Fault Tolerance: Software fault tolerance can be defined as "the ability...to detect and recover from a fault that is happening or has already happened in either the software or hardware in the system in which the software is running in order to provide service in accordance with the specification" (Inacio, 1998). The stress factor in time-pressuring situations can increase probability of faulty specifications, so that reliability and availability of DSSs is put at risk. Furthermore, users can simply avoid some development

activities because of the fear to break the application (Mackay, 1990). It is required that assistance mechanisms provide high level of fault tolerance.

The requirement is fully satisfied, if a tool is equipped with at least one facility for failure forecast, detection and recovery. It is partially satisfied, if facilities exist that cannot predict but enable recovery from eventual errors. The requirement is no prediction or recovery facilities are integrated.

T5: Efficiency: Efficiency is a measure for "resources expended in relation to the accuracy and completeness with which users achieve goals" (International Organization for Standardization, 1998). Assistance mechanisms should require minimal time, mental and financial resources to improve operational efficiency of their users (Hurwitz et al., 2005). In the context of the assistance mechanisms automation facilities are considered to be a decisive factor for efficiency of all user groups.

The requirement is fully satisfied, if a tool produces results automatically based only on the goal-based description of expected results. It is partially satisfied, if some activities to produce the results require involvement of tool operator. Finally, it is not satisfied, if most of the activities should be done manually.

2.4 Summary

This chapter described requirements on a solution of the central problem addressed by the thesis. Three guiding scenarios from emergency response, ad-hoc reporting and accommodation search domains were presented. In the scenarios two main groups of stakeholders have been identified: decision makers and software providers. The elicited requirements address the experienced problems such as lack of time for systematic development of DSSs and inability to tailor a software towards changing requirements. Finally, an assessment scheme has been

introduced that enables evaluation of technologies regarding their suitability for achieving the objectives of the thesis. The scheme is applied in the next chapter to existing methods and tools of systematic software development.

State of The Art

3

This chapter reviews state of the art of existing technologies for systematic development of Web applications. Process models and assisting tools are presented separately as they can be applied independently from each other. The selection of reviewed technologies is based on their orientation towards thesis objectives and availability of published research results within software and Web engineering communities. Based on the assessment scheme from the previous chapter, the suitability of reviewed technologies for development of DSS under time-pressure is evaluated.

3.1 Development Processes

Software development process can be defined as a structured set of activities that should be performed to develop a software product (Abran and Moore, 2004). A process defines type and order of activities, artifacts required and produced, required qualification and responsibilities of involved stakeholders. So called process models are used to provide abstractions and simplified representations of a process for particular purpose. Definition of systematic, disciplined, quantifiable development processes is the core focus of Software- and Web Engineering research disciplines (Abran and Moore, 2004; Reich et al., 2006). In the follow-

ing, three groups of approaches from these disciplines are analyzed that address objectives of this thesis from different perspectives.

3.1.1 End-User Development

End-User Development (EUD) can be defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact" (Lieberman et al., 2006). This field of research focuses on making software systems more flexible in terms of distribution of development activities, increasing their usability, understandability and learnability. A related discipline, End-User Software Engineering (EUSE), aims at improving quality of user-produced artifacts and analyzes to which extent quality-ensuring activities usually performed by professional can be performed also by end users. EUD and EUSE integrate and advance findings from other research fields such as Human-Computer Interaction, Software- and Web Engineering, Artificial Intelligence etc.

The development process utilized in the field of EUD is usually prototype-based and iterative (cf. Figure 3.1, which illustrates development process of scientific software by non-professionals). Hereby the software itself is not the goal but rather a tool to solve a situational problem (Nardi, 1993). End users prefer to build software in the try-and-error fashion instead of following a strict plan (Cao et al., 2010). Development process is mostly limited to the implementation phase as traditional planning and testing activities require programming skills that end users usually do not possess. On the other side, there is an evidence that design techniques such as drawing of mock-ups, writing simple scenarios or listing expected functionality have positive effects on the resulting solution (Rosson et al., 2007). While end users put value on correctness and reliability of created solutions, they are unwilling to invest time into systematic testing and debugging. End users seem to be overconfident regarding the correctness of their solutions (Panko, 2008). The responsibility of error detection and consistency checks is therefore put on development environments and assistance mechanisms.

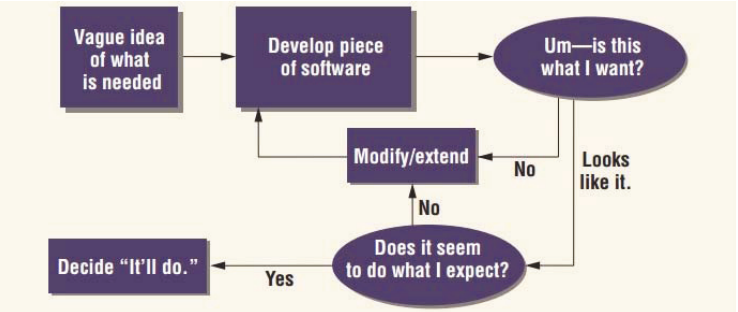


Figure 3.1.: Exploratory and Evolutionary Development of Software by Scientists (Segal and Morris, 2008)

For the implementation phase many techniques and strategies to create or to customize a software artifact have been proposed. In the following some of the most prominent techniques are presented.

Parametrization Parametrization is one of the simplest activities that end users can perform to adjust software to their needs. Although being not the primary focus of EUD it is still widely used as a first step towards more complex operations. Parametrization assumes that software is designed in a flexible way with possibility to modify its behavior according to values of a pre-defined set of parameters. At run-time software applies user-provided parameters for choice of functionality or as inputs for internal computations. An example of parametrization is configuration of toolbox layout in office suits, settings in diagramming tools or choice of encoding parameters in video cut software.

Scripting Scripting refers to the activity of creating small programs within software tools to automate manual actions (Blackwell, 2006). Utilized programming languages can be interpreted directly by the hosting tools and have different complexity. Single purpose languages use domain-oriented vocabulary and address domain experts without programming skills. The languages are used to implement keyboard

macros (e.g., Autohotkey[1]), e-mail filters (e.g., Sieve (Guenther and Showalter, 2008)) or to enhance user experience during Web browsing (e.g., Sticklet (Díaz et al., 2013)). General purpose languages are oriented towards professional programmers and can be used for a wider range of tasks. They are used to create macros in office suites (e.g., Visual Basic for Applications[2]), animations in multimedia suits (Adobe Actionscript[3]) or complete Web applications (ECMAScript (ECMA International, 2015)).

Programming-by-Demonstration (PBD) PBD is an EUD technique that enables end users to specify desired functionality by providing examples of its behavior (Blackwell, 2006). Based on demonstrated activities and data samples a PBD system tries to generalize the observed behavior and to infer a generic algorithm that can be applied to similar data or/and in the similar context. Modification of the derived algorithm is, however, not that simple as appropriate end-user-friendly representation and editing operations should be defined. PBD is applied to automate workflows (Gordon et al., 2010), text (T. Lau, 2001) and Graphical User Interface (GUI) operations (Yeh et al., 2009), to record macros in office suits (Sugiura et al., 1996) or to crawl the Web (Hartmann et al., 2007).

Visual Programming Visual Programming technique makes use of graphical representations to communicate technical concepts to end users (Burnett, 2001). End users are supposed to define behavior and structure of software by means of metaphors from a well-known domain. Visual programming doesn't require strong programming skills and in contrast to traditional text-based programming no syntax has to be learned or remembered (Burnett, 2001). The technique has been widely applied in educational environments, measurement and control systems, rapid prototyping etc. Lego Mindstorms (S. H. Kim and Jeon, 2007) and LabView (Sumathi and Surekha, 2007) are two popular programming environments utilizing the technique. Some open challenges in

[1] `http://www.autohotkey.com/`, Retrieved: 1.7.2015
[2] `https://msdn.microsoft.com/en-us/library/office/gg264383.aspx`, Retrieved: 2.7.2015
[3] `http://help.adobe.com/de_DE/as3/learn/index.html`, Retrieved: 2.7.2015

the use of this technique are fast increasing complexity of graphical representation and efficient use of available screen spaces.

Natural Language Programming (NLP) NLP refers to usage of constrained natural language to complete different tasks in software development process. The goal is to enable specification of computation or behavior as close as possible to the way how people think about them (Brad Myers et al., 2004). User input can be refined in several iterations and then transformed into internal software models. For example, this technique has been applied to infer user-task models from user descriptions of expected system behavior (Tam et al., 1998). In (Leshed et al., 2008) NLP is used to automate Web-based processes and in (Aghaee, Pautasso, and De Angeli, 2013) to create mashups. Finally, there are efforts to enable debugging and testing of applications by means of NLP (A. Ko and BA Myers, 2004).

Assessment Development processes utilized in the field of End-User Development are lightweight and focus on fast construction or modification of software artifacts. As coined by the name, EUD techniques address end users as primary target group. The latter perform development and evolution activities on their own without any assistance by professionals, so that the degree of end-user involvement is very high. Such activities as requirements engineering, design and testing from traditional software processes are omitted due to lack of time, skills and motivation by end users. The *conciseness of the process* is considered to be high. EUD processes encourage and facilitate *reuse of existing artifacts* to speed up development, to learn capabilities of utilized tools and to collect ideas for possible solutions. Making produced artifacts reusable doesn't play a major role in EUD.

3.1.2 Component-Based Development

Component-Based Development (CDB) emerged as a response to increasing complexity as well as growing costs for development, maintenance and evolution of software. One of existing definitions of the

term component describes it as a “software element that confirms to a component model and can be independently deployed and composed without modification according to a composition standard” (B. Councill and Heineman, 2001). A component model defines hereby “specific interaction and composition standards”. Development of component-based software implies “assembling components already developed and prepared for integration” (Crnković, 2003) and leads to a number of advantages including higher flexibility, lower complexity, higher quality of software as well as improved productivity of developers (A. W. Brown, 2000). The main challenge hereby is to identify, to build and to integrate pieces of functionality that are simultaneously generic enough to be used in many (possibly unknown) applications and easy enough to be understood, adapted, deployed and replaced within applications, where they have been integrated into (Crnković, 2003). Many component models, implementations and assembly strategies have been proposed. Industry widely adopted such technologies as Component Object Model (COM) (Microsoft, 2015a), .NET (Microsoft, 2015b), CORBA (OMG, 2015), Enterprise Java Beans (EJB) (I. Sun Microsystems, 2009) and OSGi (OSGi Alliance, 2015). Component-Based Software Engineering and Component-Based Web Engineering proposed many methods for systematic and disciplined development, assembly, customization and maintenance of (Web) software out of reusable components (Heineman and W. T. Councill, 2001; Gaedke and Rehse, 2000).

Existing process models for development and maintenance are usually derived from generic process models such as Waterfall (Royce, 1987), V-Model (Forsberg and Mooz, 1994), Spiral (Boehm, 1988) or Agile (Martin, 2002) and are adjusted towards systematic discovery, composition and management of reusable components. Traditional phases such as requirement elicitation, design, implementation, verification and maintenance / evolution are all present and are applied (in adapted and coordinated fashion) to both components and the software itself (cf. Figure 3.2).

The process starts with analysis of requirements for the envisioned software. Hereby possibilities of reuse of existing components are evaluated and requirements are negotiated in accordance to available components.

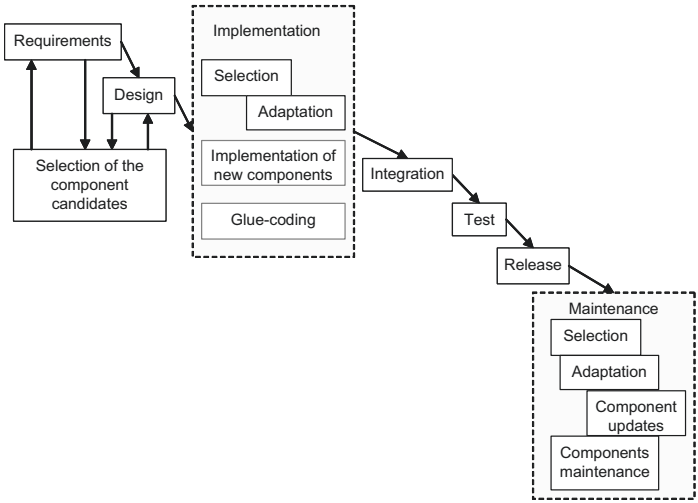


Figure 3.2.: Adaptation of Waterfall Model Towards Component-Based Development(Crnković, 2003)

During the design phase the system is decomposed into its logical units. Interfaces of components to be reused and implemented are identified. Implementation of missing components can be done using any development process but with consideration of later reuse requirements. Existing components matching system requirements are selected and adjusted towards specified interfaces (by means of parametrization, wrapping etc.). Finally, components are integrated into the overall system. Integration refers hereby to specification of connections and communication rules between components. The system is then tested against original requirements and is delivered to customer. Maintenance of component-based applications is done by replacing faulty components with new versions. Evolution of a system may require further components to be selected and added to existing design and implementation. In Component-Based Software Engineering (CBSE) and Component-Based Web Engineering (CBWE) several specialized process models have been proposed. The most mature ones are presented next.

Y Process Model The Y model (Capretz, 2005) focuses on overlapping and iteration of development activities to promote reusability of software artifacts (cf. Figure 3.3). It defines 10 phases, which can be overlapped and iterated: domain engineering (understanding the problem domain and early identification of potentially reusable components), frameworking (identification and grouping of components into domain-specific workbenches), assembly (selection of components and frameworks from specific application domains), archiving (preparation and storage of components for their later reuse), system analysis (decomposition of a software system into high-level components), design (iterative refinement of system architecture based on available components), implementation (adaptation of existing components and development of new ones), testing (verification of expected behavior of components and of the whole system), deployment (installation of the system and customer training) and maintenance (error correction and extension of a system towards new requirements). The first two phases make the Y model especially useful for development of family of systems from one application domain.

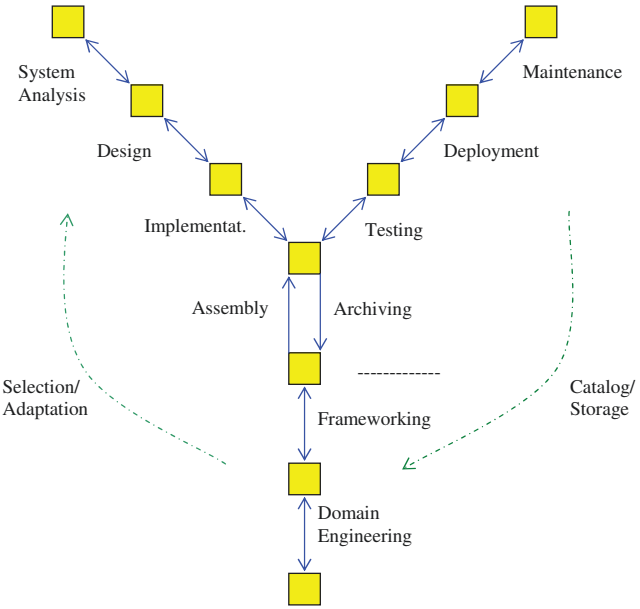


Figure 3.3.: The Y model for Component-Based Software Development(Capretz, 2005)

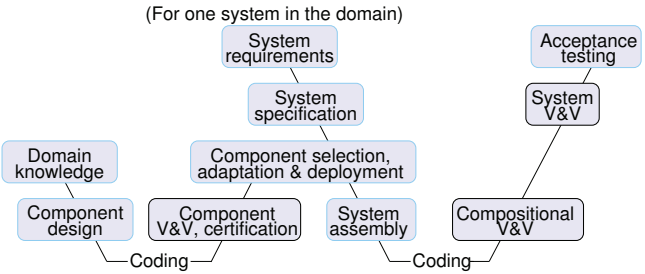


Figure 3.4.: The W model for Component-Based Software Development(K.-K. Lau et al., 2011)

W Process Model The W model (K.-K. Lau et al., 2011) puts a particular focus on Verification and Validation (V&V) of components and component-based systems (cf. Figure 3.4). The process applies V-model (Forsberg and Mooz, 1994) for their respective development and verification. Component life-cycle is explicitly described and consists of design, testing and deployment phases. The design stage foresees building reusable composition blocks based on domain requirements and publication of components in a dedicated repository. Component verification, validation and certification take place in the V&V step. In the deployment phase the component is instantiated and deployed into a target system. Development process of a component-based system is similar to the generic one described above. System design, however, takes place in form of component selection, adaptation and deployment. Together with system assembly (integration of components) system design can be executed iteratively. Compositional V&V refers to integration tests of software architecture. Finally, system V&V implies testing according to its specification.

WebComposition Process Model The WebComposition process model (Gaedke and Gräf, 2001) considers development of a component-based application as a continuous evolution. The approach refers to all reusable artifacts of software as components and provides guidance for their systematic reuse and composition. According to WebComposi-

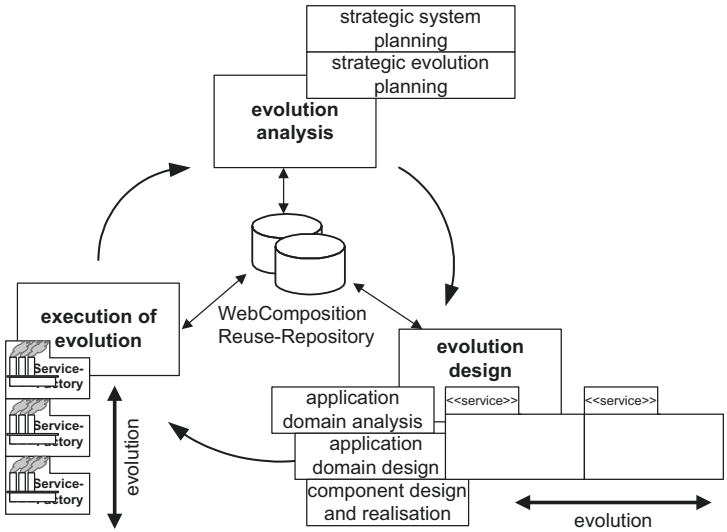


Figure 3.5.: Evolution of Web Applications Based on WebComposition Process Model (Gaedke and Gräf, 2001)

tion process model an evolution cycle consists of three phases: analysis and planning, design and realization (cf. Figure 3.5).

Analysis and planning is based on domain engineering and assumes acquisition of competence for development of a family of software within a given domain. During the design phase the gained knowledge is transformed into domain-specific models. Finally, realization phase implies selection and integration of components that implement required functionality or, if not existing, development of new ones. The approach enables application of any process models for development of single components and provides facilities for management of produced artifacts. The latter are transformed towards a shared WebComposition Model and published within a dedicated Reuse Repository. The repository provides assistance in reuse of the artifacts during subsequent evolution iterations.

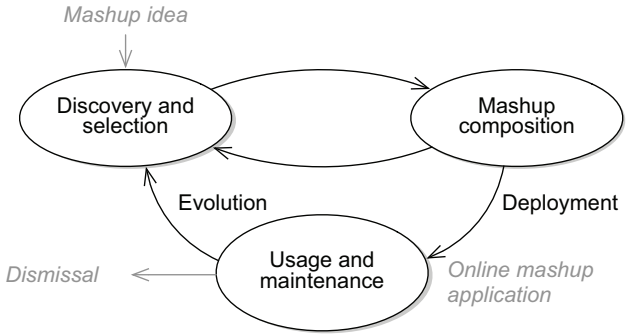


Figure 3.6.: Lifecycle Model of Web Mashups(Daniel and Matera, 2014a)

Mashup Development Recent research on so-called mashups (Yu et al., 2008) proposed a new lightweight and component-oriented development process (Daniel and Matera, 2014a) with particular focus on end users. A Web mashup is in general an application that combines "content, presentation or application functionality from disparate Web sources" (Yu et al., 2008). The process enables prototype-centric construction of situational and short-living mashups. The process consists of only three phases: 1) discovery and selection of required artifacts, 2) mashup composition and 3) usage and maintenance (cf. Figure 3.6).

In the first step of the development process mashup designers analyze and select building blocks that can be used to solve their business problem. The idea of the mashup represents hereby informal requirements on the envisioned solution. The second step implies parametrization of artifacts, definition of their coordinated behavior and configuration of application UI (depending on utilized component and composition models some activities can be skipped). Finally, the mashup is instantiated and becomes available for interaction with its users. Deployment and instantiation of mashups are usually automated by assisting tools and platforms, so that mashup designers do not even notice a switch from design to run-time phases. Maintenance phase implies activities by both mashup designers and platform providers. The former usually

repeat the first two phases to adjust existing solution towards new requirements or to test and evaluate possible implementation options. Platform providers perform maintenance of reusable software artifacts (including development of new ones) and of execution environment. For all phases the process assumes availability of appropriate tools and assistance mechanisms that should simplify handling of Web technology for non-programmers. Systematic development and maintenance of mashup components is outside of its scope, meaning that any process can be applied for this purpose.

Assessment Component-Based Development specifically focuses on systematic production, management and reuse of software artifacts, which should improve efficiency of development and maintainability of resulting solutions. CDB foresees all traditional phases of software development and additional activities related to production and management of reusable components. Although CDB processes are executed iteratively and some activities are shortened due to reuse of components, the conciseness of the overall process is average. End users are little involved into development and evolution activities – most approaches treat them as customers, with whom requirements on the system are negotiated. The exception build recently proposed mashup techniques that aim at enabling end users assemble software artifacts on their own. The overall degree of end-user involvement is considered to be low.

3.1.3 Model-Driven Development

Model-Driven Development (MDD) is a method of software development that makes use of abstract formal system specifications (models) to automatically generate software artifacts. The goal of MDD is to increase developer productivity, to improve quality and to lower costs of software. Abstract models describe structures and behavior that are similar to a family of applications or architectures and, thus, can be used to automatically generate corresponding source code. Models are in general easier to understand (also for non-programmers), faster to create and cheaper to maintain. Automatic generation of code based on abstract models reduces probability of defects, reuses programming

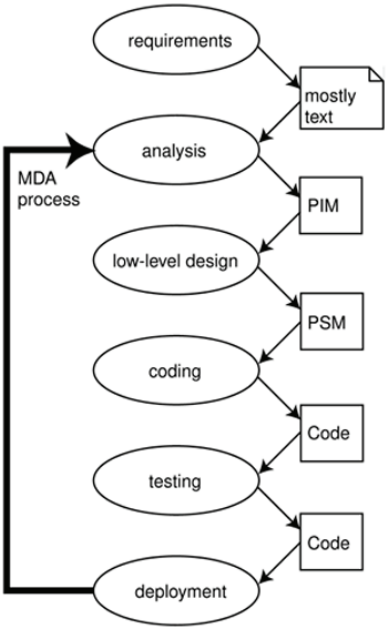


Figure 3.7.: Model-Driven Development Process (Kleppe et al., 2003)

expertise and, thus, contributes to quality of resulting software. Finally, development of software for different execution environments becomes more efficient as only environment-specific code generators have to be implemented.

MDD process foresees all traditional phases of software development and adjusts them towards continuous and systematic usage of abstract models (cf. Figure 3.7) (Kleppe et al., 2003).

The first engineering step concerns with collection of functional and non-functional requirements on the system to be developed. These should be captured in form of text and formalized in the analysis phase into so called Platform Independent Model (PIM). PIMs describe a

software system from business point of view without going into details of architectural design, implementation possibilities or possible execution environments. The process avoids production of documents and diagrams that cannot be reused in the next phases for code generation or validation. In the low-level design phase the PIM is mapped to a so called Platform Specific Model (PSM), which describes structure and behavior of a system in terms of a concrete target platform (EJB, COM, Service-Oriented Architecture (SOA) etc.). One PIM yields one or more PSMs depending on the number of platforms to be supported. Models are usually expressed in Uniform Modeling Language (UML), its extension or some Domain Specific Language (DSL). In the code phase the PSM is mapped onto constructs of a concrete programming language. Also test code to be used during system verification can be produced out of PSM. In MDD mappings from PIM to PSM and from PSM to code are performed automatically by dedicated transformation tools. Manual changes to models are possible, however, they can break logical connections between models and, thus, make later maintenance difficult. In the deployment phase the system is installed and prepared for usage by customer. Maintenance in MDD takes place in form of updates and enhancements of high-level models with subsequent automatic generation of the low-level ones. Many development approaches exist that specialize on concrete family of applications. The most adopted ones are presented below.

Web Modeling Language (WebML) WebML (Brambilla et al., 2008) focuses on model-driven development of Web applications and proposes a dedicated engineering process (cf. Figure 3.8). It applies principles of iterative development and produces partial versions of application after each cycle. The requirements phase identifies application user groups, functional requirements, information objects and so called site views specific to each user group. Conceptual modeling phase foresees design of application data and of hypertext schema (based on identified information objects and site views). Data schema is defined using Entity-Relationship diagrams or UML, while hypertext model is specified using a dedicated visual modeling language. Implementation is largely done by means of automatic code generation out of the conceptual model. Testing and validation of resulting applications is simplified due to the

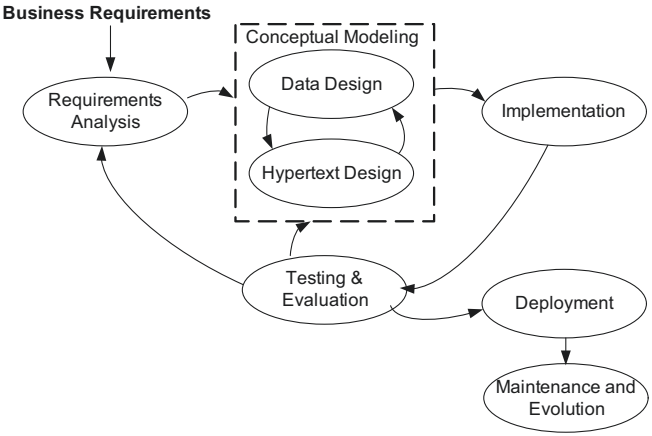


Figure 3.8.: WebML Development Process (Brambilla et al., 2008)

availability of conceptual models. Maintenance and evolution is done by repeating requirement analysis, modeling, implementation and testing phases. The process is supported by a dedicated tool WebRatio[4].

UML-based Web Engineering (UWE) A similar approach to WebML is UWE (Koch and GmbH, 2006). The main difference is that it makes extensive use of UML for modeling different aspects of a system instead of proprietary languages. UWE enables model-driven development of transaction-based, personalized, context-dependent, and asynchronous applications (Koch, Knapp, et al., 2007). It introduces several meta-models (UML profiles) to capture user requirements (UML use cases, activity diagrams or Web Requirements Engineering (WebRE) (Escalona and Koch, 2007)), structure and behavior (UWE metamodel) as well as architecture (Web Software Architecture (WebSA) (Meliá et al., 2005)) of a system. The development process starts with specification of requirements models that describe process, context and navigation-related aspects of application. The models are then transformed into several PIMs that describe content, presentation, navigation, adaptation and

[4] `http://www.webratio.com`, Retrieved: 4.7.2015

business logic of a Web application (Koch and GmbH, 2006). Different PIMs are then combined into the one that is used both for validation and PSM / code generation purposes. Due to strong UML orientation the process is supported by many existing Computer-aided Software Engineering (CASE) tools.

Semantic Hypermedia Design Method (SHDM) SHDM (Schwabe et al., 2004) makes use of Semantic Web technologies to extend expressive power of utilized models. It defines five steps for respective development process: Requirements Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation. The first step performs modeling of requirements by means of use case diagrams, scenarios and design patterns. Conceptual Design concerns with modeling of domain-related classes and relationships using ontologies. During Navigational Design possible navigation paths (nodes, links, contexts, access structures) between entities identified in the prior stage are defined. The paths are categorized according to user profiles and business tasks to be supported. In the Abstract Interface Design step navigation objects and application functionality are made available by means of abstract user interface elements. The latter are modeled using Abstract Widget Ontology and then mapped onto concrete items from Concrete Interface Widget Ontology. During Implementation the produced models are automatically transformed into executable code specific to target execution environment. The approach is supported by Synth being a dedicated modeling and code generation tool (Souza Bomfim and Schwabe, 2011).

End-User-Involving MDD There is some early work on involving end users into activities of model-driven development. In (Pérez et al., 2011) authors propose a method to increase contribution of users to MDD process. The method can be embedded into any existing MDD approach. First parts of the software that can be specified by end users are identified (as a guidance authors propose to leave quality and security-related aspects to professional programmers). Then a visual DSL is selected that should be used for end-user produced descriptions. The third step is definition of concern-specific technical models by software professionals,

which are annotated with parts to be included from end-user descriptions. Developers also provide templates and reference models to be reused by end users. Finally, end-user models are transformed and integrated into the technical ones. The process is supported with dedicated modeling and transformation tools. Development of data-intensive Web applications using visual modeling languages have been investigated in (Deufemia et al., 2013) and (Rivero et al., 2013). Authors use mock-ups and annotations to model UI, content, navigation and business logic of applications. Although the methods target non-programmers, at the time of writing no empirical evidence on applicability of the methods was available.

Assessment MDD aims in first line at improving productivity and efficiency of professional software developers. All traditional engineering phases are present, whereas implementation can be partly automated by code generators. The conciseness of the process is average. Reuse of existing artifacts in MDD is promoted due to the use of model transformation engines that accumulate knowledge and code common to a family of similar applications. However, reusability of produced models and intermediate artifacts is not explicitly addressed. Despite of some research prototypes that consider end users as creators of models, the most approaches involve them only as sources for requirements. The degree of end-user involvement is considered to be low.

3.2 Tool Assistance

Design, implementation, delivery and maintenance of software are usually supported by dedicated programs called CASE tools, which simplify or partially automate respective activities. A CASE tool can be defined as "a software component, supporting a specific task in the software-production process" (Fuggetta, 1993). The following review focuses on tools and technologies that address the thesis objectives, i.e., can be used for development of DSS and address non-professional programmers as their target group. The tools are grouped into three categories based on their common architectural properties.

3.2.1 Web Content Management Systems

With the advent of Web 2.0 participation of end users in creation of Web content significantly increased. Wikis, blog and content management software enabled even non-programmers to produce, to manage and to publish all kind of multimedia data on the Web. In general, Web content management can be defined as "the process of controlling the content to be consumed over one or more online channels through the use of specific management tools based on a core repository" (Maccomascaigh et al., 2013). By means of dedicated tools called Web Content Management Systems (WCMS) the latter can be achieved directly by domain experts without involvement of IT professionals into information management activities. WCMSs help to reduce costs for content production and publication but also for development and maintenance of data-centric Web applications. As coined by the name of these tools, their core functionality is assistance in authoring of content and related metadata. Application structure and layout are separated from the content and, thus, enable its representations in different languages and formats (incl. mobile views). Presentation and layout can be customized by so called "themes" – configurable and exchangeable artifacts, created by professional developers. Administration of WCMSs and content authoring take place over end-user-friendly UI and dedicated What You See Is What You Get (WYSIWYG) editors in a separated administration area (cf. Figure 3.9). User and rights management enable collaborative work and accounting. Some WCMSs provide native support or can be extended by means of plugins towards version control, workflow management, archiving, reporting and quality assurance functionality. Similar to themes plugins are developed by professional programmers but their integration and configuration doesn't usually require technical skills. Enterprise WCMSs also support integration with existing company IT services.

[5] Source: `https://codex.wordpress.org/images/3/30/dashboard.png`, Retrieved: 2.4.2015

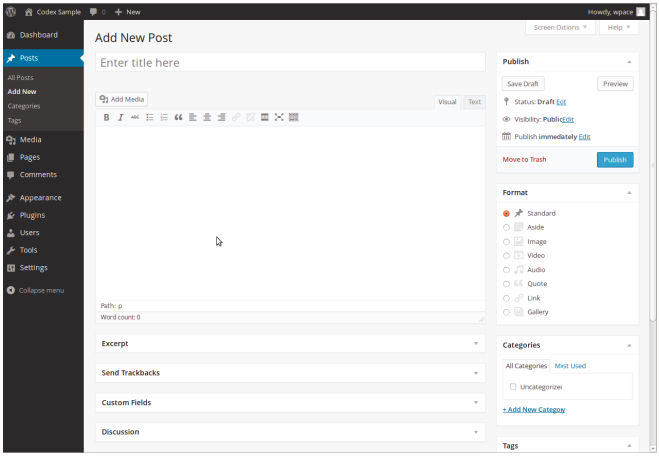


Figure 3.9.: End-User-Friendly Dashboard for Administration of WCMS[5]

According to CMSCrawler[6] and W3Tech[7] WordPress[8], Joomla![9] and Drupal[10] are the mostly installed WCMS on the Web. All of them are open-source and provide similar functionality regarding content-management. The slightly differ in their usability, performance and flexibility / extensibility of architecture.

WordPress started originally as a Web blog software and evolved to a fully-fledged WCMS. The focus of WordPress lies on usability and customizability. WordPress can be extended by numerous themes and plugins that turn the WCMS into a forum, an online shop, a social network etc. Recent research in Web Engineering proposed methods for end-user-friendly composition of plugins to achieve even better user experience (Leone et al., 2013). WordPress has a large and active community of contributors. Despite of its novice-friendliness and exten-

[6] `http://www.cmscrawler.com/tool`, Retrieved: 2.4.2015
[7] `http://w3techs.com/technologies/overview/content_management/all`, Retrieved: 2.4.2015
[8] `https://wordpress.com/`, Retrieved: 2.4.2015
[9] `http://www.joomla.org`, Retrieved: 2.4.2015
[10] `http://www.drupal.org`, Retrieved: 2.4.2015

sibility, WordPress is criticized for its performance and scalability for quickly evolving applications.

Joomla! offers similar functionality to WordPress but is more flexible in terms of configuration, customization and extension. Layout and functionality can be modified with plenty of themes and plugins. The latter enable among others fast and easy development of portals and intranets, e-commerce and social network applications. For advanced customization of Joomla some technical experience is required. For professional programmers Joomla offers a so-called Joomla! Framework[11], which consists of reusable modules for integration of external systems such as Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), data warehousing systems etc.

Drupal is a flexible, efficient and customizable WCMS. Content management capabilities are similar to those of WordPress and Joomla. A key focus of Drupal lies on collaborative authoring of structured and semi-structured content. Layout and functionality customization is achieved by means of themes and plugins. However, basic programming skills and familiarity with Web technologies are recommended for efficient use and configuration of Drupal.

According to reports by Gartner and The Forrester Wave (Maccomascaigh et al., 2013; Schadler et al., 2015) Adobe Experience Manager[12] and Sitecore Experience Platform[13] are market leaders in the field of enterprise WCMS. Enterprise WCMS provide similar content-management-related functionality but put particular focus on analytics (analysis of customer behavior and site effectiveness), marketing automation (spreading content over variety of communication channels such as the Web, mobile devices or social media), personalization (view and content adaptation based on customer profiles) and e-commerce (integration with enterprise services, data and workflows).

[11] `http://framework.joomla.org/`, Retrieved: 2.4.2015

[12] `http://www.adobe.com/de/solutions/web-experience-management.html`, Retrieved: 2.4.2015

[13] `https://www.sitecore.net/platform.aspx`, Retrieved: 2.4.2015

Assessment WCMS enable non-programmers to easily create data-intensive Web applications and manage the corresponding content. The content itself is meant to be consumed by other people. WCMS do not focus on customizable integration of heterogeneous data sources and functionalities. The latter is, however, possible by means of plugins created by professional developers. Programmers can wrap any kind of data source or functionality as a plugin and offer it for integration within a WCMS system. Despite some early research on flexible plugin composition methods (Leone et al., 2013), most of existing WCMS support visual aggregation of plugins only (i.e., without any orchestration or choreography logic). Configuration of content layout and presentation is possible within boundaries of the selected theme. Regarding evolution assistance, WCMS do not offer any automation tools to develop plugins, so that new data-sources and functionalities have to wrapped manually. Due to missing integration logic, plugins can be freely combined with existing ones. Once developed, plugins can be published and discovered from local or global plugin repositories. The repositories automatically publish plugin metadata and enable their efficient storage and discovery. WCMSs address non-technical users (acting as WCMS administrators) as primary target group and use vocabulary and operation techniques familiar to most of Web users. No programming skills are required to operate a WCMS. WCMSs allow administrators to perform a limited and foreseen set of configurations, i.e., faulty ones can be early detected and prevented by input validation. Configuration assistance mechanisms show warnings on implications potentially harmful configuration parameters. The degree of fault-tolerance is considered to be high. Automation in WCMS is average – they automate internal processes required to store and display Web content, but need manual design and configuration of precise definition of the desired results on the UI level.

3.2.2 Dashboards and Ad-Hoc Reporting Tools

Dashboards (also known as MIS, Executive Information System (EIS) or portals) and ad-hoc reporting tools belong to the class of so-called Business Intelligence software. BI refers usually to technologies that enable

"collection and preparation of data about company and its environment as well as its visualization in form of business-related information for analysis, planning or control" (Gluchowski et al., 2008). Dashboards and ad-hoc reports focus on interactive and flexible visualization of distributed company data. In contrast to other BI tools, dashboards and ad-hoc reporting tools are highly customizable and do not require technical knowledge to be used by end users. As a result, they can be used for development of data-driven DSS by end users.

Dashboards aim at aggregating business-related data on one screen and providing compressed but expressive views on it. Dashboards make extensive use of specialized visualizations such as tachometers, progress bars, charts, maps etc. Usually they are tailored towards the needs of a single person or group of people, which leads to differentiation between executive, tactical and operative dashboards (Gluchowski et al., 2008). Executive dashboards target business executives and support long-term strategic planning. They focus on compressed performance indicators and intuitive visualization instead of on rich interaction possibilities. Tactical dashboards target leaders of department (financial, sales, human resources etc.) and support analytical processes that can help to identify trends or to find problem causes. Operative dashboards target project managers and knowledge workers, who need to monitor different kind of processes or activities and have to quickly react in case of exceptional situations. The latter is supported by integration of process control or management functionality into the dashboard. In general Hurwitz & Associates identify three functionalities that bring the greatest value to dashboard users: linking historical business information with real-time operational data, alerting stakeholders in case metrics or indicators are out of tolerance and enabling connectivity to messaging systems and business processes (Hurwitz et al., 2005).

Dashboards are usually implemented using dedicated software that enables integration of multiple heterogeneous applications within a unified user interface (cf. Figure 3.10). In addition to data access applications frequently performed office activities such as E-mail, voice or chat communication, contact or calendar management can be integrated. Single Sign On is considered to be one of the most important

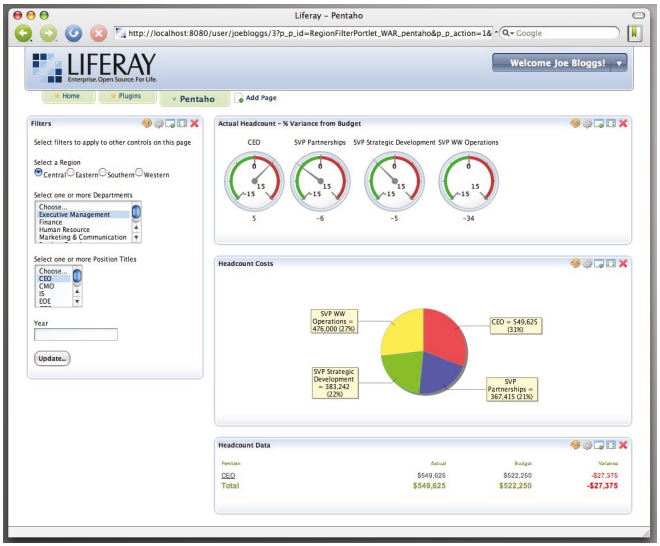


Figure 3.10.: Example of a Dashboard Implemented Using Liferay[14]

features of dashboards and ensures that users can quickly authenticate themselves and get immediate access to all relevant resources. Dashboards provide rich personalization facilities. First, functionality and layout can be adapted based on user group or role. Second, users can perform individual personalization, e.g., by choosing applications to be integrated, changing their layout or display mode and modifying application-specific preferences. Some dashboards support collaboration in form of message-, voice- or video chats as well as forums or group workflows.

Liferay[15] and IBM WebSphere Portal[16] are examples of tools to build Web portals and dashboards. They rely on Java Portlet specifications (Sun Microsystems, 2003) (Sun Microsystems, 2009) and enable application

[14]Source: `https://www.liferay.com/de/community/wiki/-/wiki/Main/Pentaho+Portlets`, Retrieved: 13.4.2015
[15]`http://www.liferay.com`, Retrieved: 10.4.2015
[16]`http://www-01.ibm.com/software/de/websphere/`, Retrieved: 10.4.2015

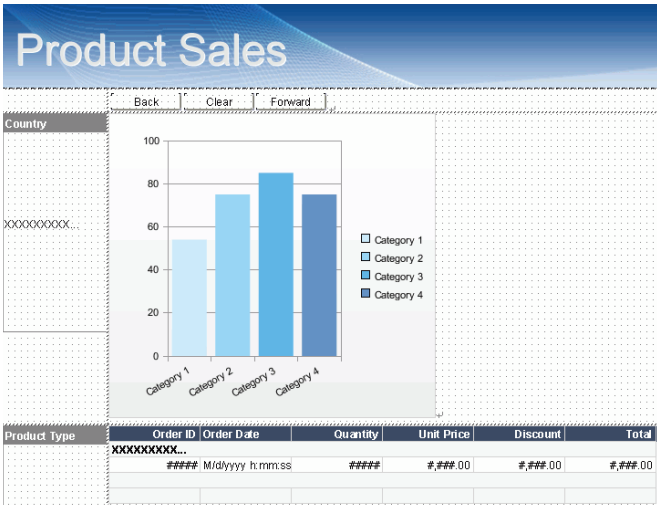


Figure 3.11.: Specification of Report Layout in JReport[17]

integration on the presentation level. UI components called portlets are fully-fledged Web applications that can be developed by professionals and are prepared for inclusion as well as parametrization within Java portals. Portlets wrap internal and external data sources or provide access to different Application Programming Interfaces (APIs). Portlets can interact with each other using different mechanisms of inter-portlet communication, which are, however, not configurable by end users.

Ad-hoc reporting tools aim at assisting analysts, knowledge workers, executives and managers in fast collection and visualization of business-related data. The goal is to fill a gap between reports created by professional developers for a fixed set of use cases and situational information needs of end users. Furthermore, results should be produced faster than in case of delegation to IT departments. The tools provide intuitive UIs and guide users during report definition process. Usually, report specifications consist of data sources to be aggregated, filters, grouping

[17] Source: `http://www.jinfonet.com/kbase/jreport13/tutorial/index.htm`, Retrieved: 15.4.2015

and sorting parameters as well as layout and visualization settings (cf. Figure 3.11). Some solutions offer visual environments for construction of queries or for definition of new layouts. Resulting reports can be both simple data tables and charts and complex interactive solutions such as dashboards. Examples of ad-hoc reporting tools with the described functionality are JReport[18], Style Report Enterprise[19], or Ubiq Reports[20].

Assessment Dashboards and ad-hoc reporting tools provide intuitive UIs and interaction concepts for customization and adaptation. Building blocks (data source adapters, visualization templates, widgets) are developed by professional developers and can provide any data, logic or presentation functionality. Reporting tools comprise data and presentation artifacts only. Freedom of data source / functionality selection is considered to be high. However, users are not able to specify any additional behavior on top of the aggregated components, so that flexibility in definition of integration logic is low. User interface can be customized based on developer-provided themes and layouts. Dashboards and ad-hoc reporting tools provide little to no assistance for evolution of resulting solutions. New data sources or layouts have to be wrapped and prepared for inclusion manually. The same holds for adaptation towards interoperability and composability. Internal registries, however, enable efficient management and discovery of reusable functionality. The overall evolution support of dashboards and ad-hoc reporting tools is considered to be average. Non-programmers is the primary target group of this class of tools. Most activities are done using wizards or simple drag-and-drop operations, so that complexity of utilized abstractions and interaction patterns is low. Dashboards and ad-hoc reporting tools prevent data corruption and crashes of run-time environment by timely analyzing user-entered parameters and issuing warning on faulty configurations. The degree of fault-tolerance is high. Automation facilities are scarce, as users have to define all aspects of the desired solutions manually.

[18] `http://www.logianalytics.com/index.php?q=ad-hoc-reporting-overview`, Retrieved: 15.4.2015
[19] `https://www.inetsoft.com/products/StyleReportEE`, Retrieved: 15.4.2015
[20] `http://ubiq.co`, Retrieved: 15.4.2015

3.2.3 Composition Tools

Composition tools facilitate end-user development of software by simplifying knowledge and code reuse. They enable construction or modification of software using well-defined configurable components acting as basic building blocks. The latter are usually provided by professional programmers and have a goal to hide complexity of underlying technology behind end-user-friendly UI or intuitive visual metaphors. To develop a solution out of given building blocks end users have to devise a composition plan that would specify blocks to be used, their respective configuration and mutual relationships. The process of discovery and composition is usually supported by dedicated artifact repositories and plan integrity check algorithms. To facilitate and speed up the development process composition tools offer possibilities to reuse knowledge and experience of other developers. While dedicated recommendation mechanisms suggest artifacts or configurations to complete partial compositions, (real-time) collaboration facilities enable joint development of solutions by a group of geographically distributed people.

Many research and commercial tools with focus on end-user-friendly development of data-intensive Web applications have been proposed. Microsoft FrontPage[21] and Macromedia Dreamviewer[22] were the first tools with WYSIWYG capabilities. At the beginning they focused on development of static Web sites, but were quickly enhanced towards support of interactive, data-driven applications. The tools enabled also integration of XML-based sources such as Web services, so that users could construct simple portal applications aggregating data from different sources. Click (Rode et al., 2005) and XIDE (Litvinova et al., 2012) are research prototypes that applied findings from HCI, EUD and Web Engineering fields to devise end-user-friendly composition environments. The tools provide a library of reusable software artifacts that can be combined and configured towards interactive data-driven Web applications. The artifacts have different level of granularity that were devised according to identified user mental models. Click and

[21] `https://msdn.microsoft.com/en-us/library/aa167865%28v=office.11%29.aspx`, Retrieved: 8.4.2015

[22] `http://www.adobe.com/de/products/dreamweaver.html`, Retrieved: 8.4.2015

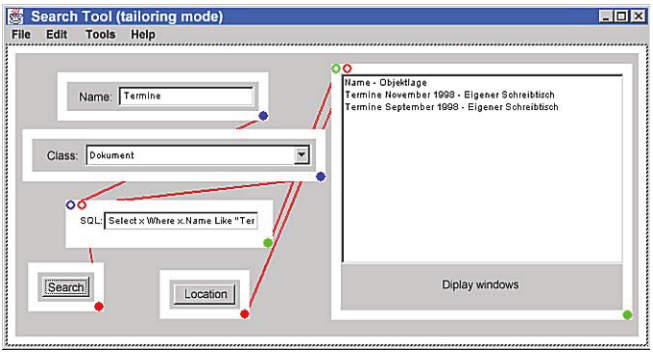


Figure 3.12.: A Platform for End-User Tailoring of Component-Based Software (Source: (Mørch et al., 2004))

XIDE do not distinguish between design and runtime phases - user-performed configurations become immediately visible. The tools offer guidance through development process, consistency checks and support for collaborative activities.

A composition tool that strictly applies principles of component-based software development has been developed in (Won et al., 2006) (cf. Figure 3.12). Authors propose a component model and a development platform that should enable end-user tailoring of component-based software. The so-called FlexiBeans model considers basic building blocks as black boxes, whose interfaces consist of named event ports and shared variables. These so-called atomic components can be composed to more complex and intelligent structures, and, thus, simplify overall application design. The FreEvolve platform enables discovery, configuration and run-time re-composition of components by means of the "wiring" metaphor. It provides multiple views (2D, 3D and multi-windowed 2D) on composition plans and devises a number of HCI techniques to simplify the tailoring process. To avoid faulty configurations that can break the overall application the platform performs rule-based integrity checks of composition plans. Furthermore, groupware behavior of a tailored system can be tested within a dedicated exploration environ-

ment. Collaboration between different users and exchange of artifacts is supported by means of a specialized component repository.

With the advent of SOA new technologies and methods for software development emerged. SOA promotes decomposition of software into loosely-coupled standardized building blocks, so that the latter can be efficiently maintained and reused over time. In SOA basic business-level functionalities are wrapped into so called "services" and are published within one or several directories. Complex business processes are then realized as a design- or run-time composition of services. Systematic production, discovery and reuse of services make development and maintenance of software more time- and cost-efficient. One of the most prominent implementations of SOA is the Web Services Architecture (WSA) (Booth et al., 2004). Its primary goal is to promote interoperability and compatibility between heterogeneous platforms and programming languages. The elementary building block of WSA is a Web Service, which is a "software system designed to support interoperable machine-to-machine interaction over a network" (Haas and A. Brown, 2004). Interface of a Web service is described using a machine-processable format such as Web Service Description Language (WSDL) and defines its functional and non-functional properties. According to (Booth et al., 2004) communication with Web services takes place using SOAP messages, which are typically transferred over Hypertext Transfer Protocol (HTTP). Composition of Web Services towards business processes can be declaratively described by XML-based Business Process Execution Language (BPEL) (Alves et al., 2007). Graphical BPEL editors provide environments for visual composition and execution of Web Services (e.g., BPEL Designer Project[23] or Oracle BPEL Process Manager[24]). Beside WS*-Technologies other implementations of SOA exist. So called Resource-Oriented Architecture (ROA)s (Lucchi et al., 2008) rely on Representational State Transfer (REST)ful Web services (Fielding, 2000) instead of SOAP-based ones. In (Daniel, Soi, et al., 2011) authors describe an implementation of SOA using graphical components. While SOA composition tools can simplify and speed-up orchestration of dis-

[23] `https://eclipse.org/bpel/`, Retrieved: 9.4.2015

[24] `http://www.oracle.com/technetwork/middleware/bpel/overview/index.html`, Retrieved: 9.4.2015

tributed functionalities, they still address professional developers and require deep understanding of underlying principles.

Composition tools addressing non-programmers have been developed in the context of Web mashups. Mashups offer additional functionality on top of the aggregated components in form of customizable views, logic or novel interaction possibilities. Depending on the type of combined components one distinguishes between data, application logic, UI and hybrid mashups (Daniel and Matera, 2014b). *Data mashups* retrieve, transform, combine and visualize heterogeneous Web data sources. The latter are typically Really Simple Syndication (RSS) (Winer, 2003) or Atom (Nottingham and Sayre, 2005) feeds, Hypertext Markup Language (HTML) pages or RESTful services. *Logic mashups* combine distributed functionalities such as Javascript API or Web services to interactive applications or automated processes. *Presentation mashups* aggregate visual components such as Javascript UI widgets, portlets (Sun Microsystems, 2003), World Wide Web Consortium (W3C) widgets (Cáceres, 2012) or OpenSocial gadgets (OpenSocial and Gadgets Specification Group, 2014) and define synchronization logic on top of them. Finally, *hybrid mashups* combine different types of components to full-fledged Web applications or services. So-called mashup editors (or platforms) have been proposed to simplify development and execution of mashups. Usually they target non-programmers and focus on simplicity and usability. Based on findings in the EUD and HCI fields mashup editors enable development of mashups using visual or natural programming environments. In the following, several commercial and research tools focusing on different types of mashups are presented. The selection is based on the latest published developments accessible on the Web.

Yahoo Pipes[25], DERI Pipes[26] and MyCocktail[27] are examples of editors for development of data mashups. Yahoo Pipes enables graphical definition of data flows based on selected data sources, operators and connectors (cf. Figure 3.13). The tool can access any XML-, JSON- or

[25] `http://pipes.yahoo.com`, Retrieved: 11.4.2015
[26] `http://pipes.deri.org`, Retrieved: 11.4.2015
[27] `http://sourceforge.net/projects/mycocktail`, Retrieved: 11.4.2015

CSV-based data sources and combine them into a so-called “pipe”. Different operators enable aggregation, filtering and transformation of data. Outputs of one operators or sources can be graphically mapped onto inputs of others. The result of pipe execution is a new data feed, which can be exported in RSS, Atom or Javascript Object Notation (JSON) format. MyCocktail enables similar functionality to Yahoo Pipes. However, it is open-source and supports RESTful services described by Web Application Description Language (WADL) (M. J. Hadley, 2009). Heterogeneous data sources can be combined into one by means of different operators. The output is visualized using one of so-called renderers, which are essentially HTML templates for different purposes. The tool can render interactive tables, lists, maps, charts etc. and export the result as a iGoogle or Netvibes gadget or W3C widget. Finally, several rendered mashups can be placed on one canvas and exported as a Web application. DERI Pipes also supports composition of Extensible Markup Language (XML)-and JSON-based data sources but focuses on semantic data sources such as Resource Description Framework (RDF) or microformats. Provided operators enable among others querying of data using SPARQL Protocol And RDF Query Language (SPARQL) and XQuery (Robie et al., 2011). Programmers can also set of available operators with custom ones. Pipes can be executed locally using a command-line tool or integrated into Web applications. Although the tools claim that even non-programmers can use them and create useful mashups, the complexity of data transformation task and the need of basic programming skills didn’t bring them the expected popularity. As a response to this observation, research community developed end-user-oriented recommendation mechanisms and domain-specific mashup tools. For example, Baya (Roy Chowdhury, Rodríguez, et al., 2012) is a browser extension that enriches functionality of Yahoo Pipes and MyCocktail. It makes use of collaborative-filtering techniques and suggest possible mashup configurations based on prior community experience. Baya recommends operator parameters or module connections and automatically applies them on users will. ResEval (Imran et al., 2012) is an example of a domain-specific mashup. The editor focuses on concepts and terminology from one particular domain (research evaluation) and avoids tasks related to data mediation. The interface is also specialized towards the domain and uses concrete visual metaphors

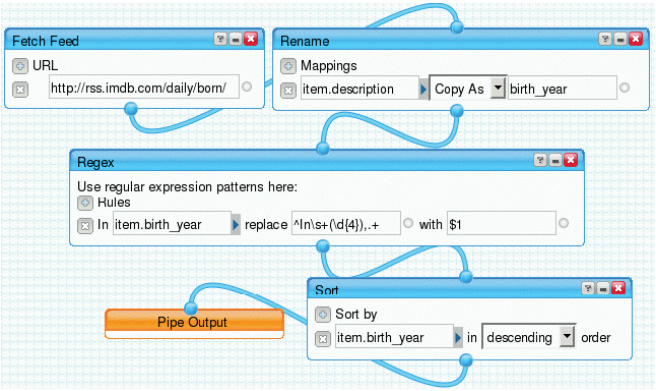


Figure 3.13.: Yahoo Pipes: a Data Mashup Editor[28]

instead of generic "data source" or "operator" blocks. First user studies indicated that domain-orientation and composition assistance can significantly simplify mashup development.

SimpleBPEL (Juhnke et al., 2010), ServFace Builder (Nestler et al., 2011) and Widget Composition Platform (Spahn and Wulf, 2009) focus on logic mashups and enable end-user-friendly composition of Web services. SimpleBPEL is a tool that enables composition of workflows based on domain-oriented BPEL fragments. The latter are identified, annotated and prepared for reuse by BPEL experts. Domain experts can then graphically connect inputs and outputs exposed by the fragments. The tool validates the composition, generates required mediation code and exports results as a valid BPEL workflow. ServFace Builder makes use of a dedicated annotation language to generate form-based UIs out of Web service operations. The tool addresses non-programmers and enables them to combine different Web services by defining data flow in the WYSIWYG fashion (cf. Figure 3.14). Service compositions can be placed within so-called pages and corresponding navigation behavior can be defined. A similar approach has been proposed by Spahn and

[28] Source: `http://pipes.yahoo.com/pipes/docs?doc=operators`, Retrieved: 13.4.2015

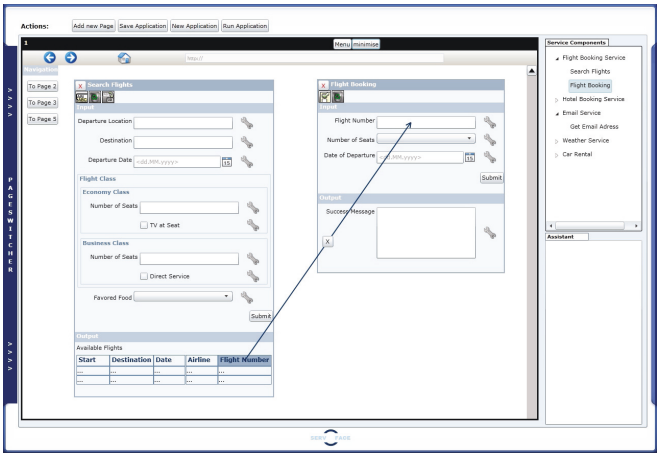


Figure 3.14.: ServFace Builder: a WYSIWYG Service Composition environment(Nestler et al., 2011))

Wulf in (Spahn and Wulf, 2009). Authors describe a tool for graphical composition of enterprise resources such as ERP and CRM systems, custom software and external resources. Each resource should be prepared for composition by means of dedicated wrappers that unify their interfaces. Iteratively users can put services to be combined on a common canvas and visually define connections between their inputs and outputs. Compositions are then wrapped into widgets, i.e., "small, self-contained, interactive applications" that can be executed within Yahoo! Widget Engine[29] or Microsoft Windows Vista Sidebar[30]. Some research prototypes enable assisted and even automatic composition of services. For example, in (Xiao et al., 2010) authors describe a framework to generate Web service-based processes based on given keywords and domain-specific ontologies. The latter describe annotated task models that together with historical service usage data help to discover services required for composition. Users can personalize constructed plans and

[29] `https://info.yahoo.com/privacy/us/yahoo/widgets/details.html`, Retrieved: 10.4.2015

[30] `http://windows.microsoft.com/en-us/windows-vista/windows-sidebar-and-gadgets-overview`, Retrieved: 10.4.2015

add replace automatically assigned services. The tool, however, doesn't consider syntactic and semantic mismatch between service interfaces, so that outputs have to be mapped on inputs manually by end users. In (Mehandjiev, Lécué, et al., 2010) authors propose to apply semantic technologies and formal logic to describe functional and non-functional properties of Web services. End users can choose between task-based templates organized in a domain taxonomy and the tool automatically discovers compatible implementations. Users can accept proposed assignments or choose alternative services that are recommended based on their functional compatibility.

Simple presentation mashups can be developed using tools like Netvibes[31] or Apache Rave[32]. Netvibes enables aggregation of several UI components called *apps* on one page. An app can be either manually developed by professional developers or automatically generated for given RSS feeds. Users can personalize apps by modifying exposed attributes. Netvibes doesn't allow any logic definition on top of the apps, which affects overall usability and usefulness of produced mashups. Apache Rave also enables composition of UI components on one screen but applies other component technologies such as W3C widgets (Cáceres, 2012) and OpenSocial gadgets (OpenSocial and Gadgets Specification Group, 2014) and offers infrastructure for interactions between components. Development of mashups in Apache Rave is restricted to selection, placement and personalization of UI components. Logic on top of components cannot be defined by end users but rather emerges in the self-organized fashion based on component capabilities.

Presentation mashups can be developed using such tools as Platform for End User Development Of Mashups (PEUDOM) (Matera et al., 2013) or Engineering of Do-it-Yourself Rich Internet Applications (EDYRA) (Rümpel et al., 2011). PEUDOM is a mashup environment that enables visual composition of UI components called widgets (cf. Figure 3.15). The latter follow a proprietary component model and can be developed within

[31] `http://www.netvibes.com/en`, Retrieved: 10.4.2015
[32] `http://rave.apache.org`, Retrieved: 13.4.2015
[33] Source: `http://de.slideshare.net/matteo_picozzi/dashmash-a-mashup-environment-for-end-user-development-8396506`, Retrieved: 13.4.2015

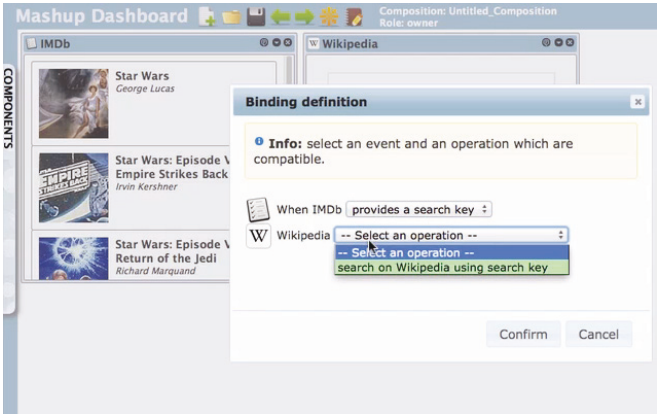


Figure 3.15.: PEUDOM: a Visual Environment for Development of UI Mashups[33]

the platform itself. To develop a widget users have to define a mapping between outputs of some RESTful service or RSS feed and a pre-defined visualization template. Ready widgets can be freely aligned within the composition canvas and interact with each other using Inter-Widget Communication (IWC). The latter is specified by users as mapping between events of one widget and operations of another. The platform is equipped with real-time collaboration facilities to enable concurrent development of mashups by groups of users. EDYRA builds on top of the CRUISE composition platform (Pietschmann, Radeck, et al., 2011). EDYRA enables composition of UI components with specification of their communication behavior. Functional and non-functional properties of components are semantically described, so that the platform can utilize these descriptions to automatically mediate, discover or recommend partial mashup configurations. Dedicated views provide explanations to component capabilities and connections in natural language (Radeck et al., 2013).

There are some tools that enable development of hybrid mashups. For example, NaturalMash (Aghaee and Pautasso, 2014) can combine data source, APIs and presentation elements into one Web application. The

composition takes place using controlled natural language, i.e., "natural language restricted in terms of vocabulary and grammar", and in the WYSIWYG fashion. The tool accompanies text-based specification with interactive visual environment. Mashup specifications are constantly evaluated by a compiler that gives immediate feedback on composition results or eventual errors to users. JackBe Presto[34] is a commercial tool that enables composition of data sources, Web services and UI components. Data mashups are developed in the pipe fashion similar to Yahoo Pipes. Results can be visualized using UI components called *apps* and then be aggregated on once canvas and synchronized by means of IWC. A similar approach has been investigated within the EzWeb/FAST (Lizcano et al., 2008) and MashArt platforms (Daniel and Matera, 2009). Development of the both has been discontinued, however.

Assessment Most of the tools enable integration of any data source / functionality if it is wrapped according to the respective component model. The freedom of selection is therefore high. Logic defined on top of the components is usually restricted by the composition model and interfaces of aggregated components. Configuration of UI of composite applications is possible by interactive placement of components and their parametrization. The overall development assistance is average. Many composition tools provide mechanism to support evolution of user-created solutions. Integration of new components is facilitated by semi-automatic transformation engines (cf. generation of widgets out of Web services in MyCocktail and PEUDOM). Updates to components and interoperability improvements are usually performed manually. Composition tools usually operate dedicated repositories for collection, discovery and sharing of reusable artifacts and of complete compositions. Composition tools are equipped with interfaces that can be easily learned and operated by non-programmers. Visual metaphors, controlled natural language and domain-orientation help to lower overall complexity of the tools. Nevertheless, basic understanding of programming activities such as definition of control flow between components are required to create solutions for real-world problems. The complexity is considered to be average. User configurations and compositions are

[34] `http://jackbe.com/products/presto`, Retrieved: 13.4.2015

usually checked against syntactic or semantic mismatches so that many runtime errors can be prevented before they actually occur. Degree of fault-tolerance is therefore high. Automation in composition tools have been a topic of research for many years. Some prototypes are able to automatically recommend partial compositions, to discover potential components for a given task or to create complete solutions based on formal goal descriptions. However, development using composition tools remains mostly manual process – mostly due to the missing facilities for end-user-friendly goal specification.

Table 3.1 summarizes the assessment of state-of-the-art technologies for development and evolution of DSSs.

3.3 Discussion

As shown in the previous section suitable solutions exist to increase involvement of end users into development process, to improve maintainability of produced artifacts or to automate programming activities. However, no holistic approach exists that would address these aspects simultaneously and, thus, enable end users to develop maintainable software under time-pressure. Each reviewed technology has its weaknesses with regard to at least one requirement from Chapter 2 – a suitable combination or modification of technologies to fulfill all of them has not been developed yet.

Techniques from the EUD group focus on maximal end-user involvement and score best in the comparison with other groups. The utilized processes are short and produce working results quickly. However, due to insufficient quality ensuring activities maintainability and reusability of produced solutions is rather low. Component-based methods enable the highest reuse and, thus, time- and cost-efficient evolution of produced artifacts. Their primary target group is, however, professional software developers. Application of these methods by decision makers is therefore limited. The same is true for MDD technologies – they focus on development of maintainable products, but can require program-

Table 3.1.: Comparison of Analyzed State-of-the-Art Technologies. Labeling: ++ Requirement fully satisfied, + Requirement partially satisfied, − Requirement not satisfied, / not applicable

Requirement \ Technology	End-User Development	Component-based Development	Model-Driven Development	Web Content Management Systems	Dashboards and Ad-Hoc Reporting Tools	Composition Tools
End-User Involvement	++	−	−	/	/	/
Reuse-orientation	+	++	+	/	/	/
Process Conciseness	++	+	+	/	/	/
Development Assistance	/	/	/	+	+	+
Evolution Assistance	/	/	/	+	+	+
Ease of Use	/	/	/	++	++	+
Fault Tolerance	/	/	/	++	++	++
Automation	/	/	/	+	+	+

ming skills to be applied. All of the presented assistance tools can be applied by decision makers to produce DSSs and customize them as soon as requirements change. User-friendly interfaces and facilities to prevent faulty specifications are provided. However, to increase usability, tools often operate with abstractions that impede their own expressibility and limit functionality of the resulting DSSs. Deficiencies also exist in the quality of automation facilities, resulting in time-consuming configuration steps being performed manually.

The identified deficiencies make application of the reviewed technologies under time-pressure inadequate. To provide an efficient assistance, a holistic approach is required that would take all the elicited requirements into account.

3.4 Summary

This chapter presented state-of-the-art technologies related to the objectives of the thesis. Two categories of approaches has been introduced: development processes and tool assistance. Their assessment unveiled the lack of a holistic solution that would combine a concise, end-user-involving process, reuse-orientation and efficient assistance mechanisms. Findings of this chapter build input for the solution concept presented next.

4 WebComposition for End-User Development Approach

This chapter presents an approach called WebComposition for End-User Development. Section 4.1 gives an overview of the approach components and describes their relationships. Section 4.2 introduces the design principles for realization of the approach. Then conceptual models and a process for development of DSSs under time-pressure are presented (cf. Section 4.3 and Section 4.4). Finally, assisting mechanisms for development and evolution of produced solutions are introduced (cf. Section 4.6).

4.1 Overview

The WebComposition for End-User Development (WebComposition/EUD) approach proposes a method for systematic development of DSS under time-pressure. Its structure is based on design measures proposed in (Wallmüller, 2001), which have a goal to ensure quality of both the

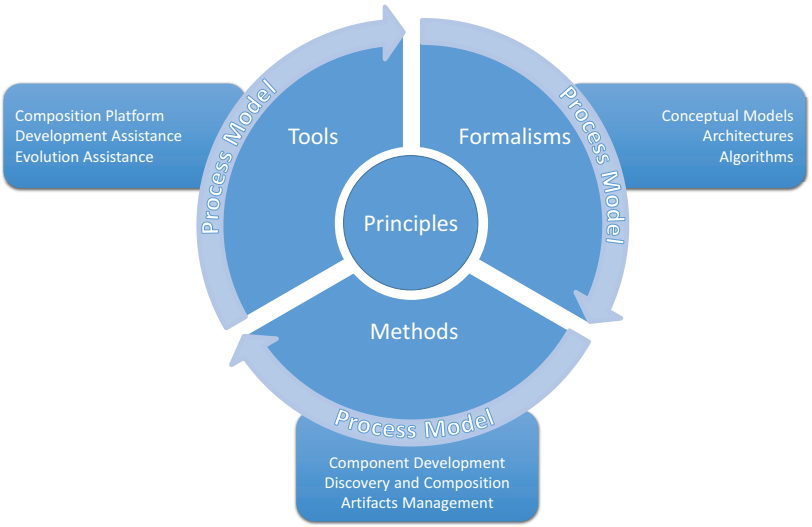


Figure 4.1.: Quality-ensuring Design Measures of the WebComposition/EUD Approach (Based on (Wallmüller, 2001))

software development process and of the produced results (cf. Figure 4.1).

The core of the approach are quality-ensuring *Principles*, which in addition to the identified requirements drive design decisions for definition of other components. The *Formalisms* provide theoretical basis for concepts used in the approach. They include conceptual models of elementary building blocks, compositions, interaction algorithms and architectures of implementing platform. The *Process Model* describes phases, artifacts, roles and responsibilities, required for systematic and efficient development of software products under time-pressure. The *Methods* are techniques, which are applied during various phases of the Process Model to produce artifacts in time- and cost-efficient fashion. WebComposition/EUD proposes methods for development and evolution of model-compliant components, for dialog-based discovery and for management of heterogeneous artifacts. Different steps of the

proposed process are assisted by *Tools* that aim at simplifying and automating time-consuming development activities. The main tool of the WebComposition/EUD approach is so called Composition Platform with integrated assistance mechanisms. Other tools targets evolution phase of the process and provide assistance for maintenance and extension of end-user-built solutions.

4.2 Principles

The following best practices from EUD, HCI and Web engineering research fields (Lieberman et al., 2006; C. M. Brown, 1988; Wilde and Gaedke, 2008) lead to several principles and design guidelines that should increase effectiveness and efficiency of the devised approach.

Gentle Slope of Complexity A system should expose its capabilities and flexibility according to skills of its users. Novices should not be overwhelmed with complexity of development process but rather start with simple or familiar concepts and gradually learn the development environment (Repenning and Ioannidou, 2006). An optimal balance between posed challenges and user skills results in efficient learning process and helps to avoid anxiety or boredom (Csikszentmihalyi, 2008).

Live programming The principle of "liveliness" promotes smooth translation between the phases of software design (modeling or programming) and execution (Tanimoto, 1990; Aghaee and Pautasso, 2013). The goal is to foster understanding and evaluation of system state by providing immediate feedback on performed changes without need to rebuild the system or switch to alternative views. Immediate feedback helps to spot relationships between system modeling constructs and their outputs more easily. Furthermore, sources and causes of errors can be identified faster.

Domain Orientation A development environment should support activities from and apply terminology of end-user problem domain

(Nardi, 1993). Domain orientation avoids necessity of learning generalized concepts and doesn't require users to map their problems on artificial constructs (Repenning and Ioannidou, 2006). Although domain-orientation decreases generality of tools and shrinks their application areas, the usability and efficiency of tools increases (Imran et al., 2012).

Separation of Responsibilities The principle promotes an appropriate balance between development activities of end users and of professional programmers. Tasks that require programming skills, knowledge of or access to the whole IT environment, should be assigned to stuff with corresponding education and responsibilities (Nardi, 1993). As a result professional developers can better control and optimize end-user solutions towards efficiency, security and reliability (G. Fischer et al., 2004).

Knowledge and Artifact Reuse Sharing of knowledge and software artifacts is a widely applied practice in the field of EUD (Nardi, 1993). Novice users should be able to reuse and to learn from artifacts developed by experienced peers (Wiedenbeck, 2005). Solutions for common problems should be shared to avoid repetitive and time-consuming developments. Furthermore, building a new software by modifying an existing one can speed up the process and improve quality of results.

Loose Coupling The principle promotes agility in software architectures, so that the software can adapt to changes and evolve more efficiently (Kaye, 2003). Loose coupling facilitates reuse of software artifacts and decreases effort required to integrate new functionalities, particularly important in distributed Web applications (Wilde and Gaedke, 2008). Decoupling of software components can be achieved by minimizing their mutual dependencies, introducing asynchronous document-style messaging, late binding, standardizing interfaces etc. (Kaye, 2003).

4.3 Formalisms

The basic idea of the WebComposition/EUD development approach is to wrap heterogeneous data sources and functionalities into uniform building blocks and to enable their "end-user-friendly" composition into integrated solutions. Conceptual models and core algorithms presented in this section provide a theoretical basis for the approach. The conceptual architecture of enabling platform is discussed in Chapter 5.

4.3.1 Component Model

WebComposition/EUD components build up on the following definition of a software component:

Definition 4.3.1. A *software component* is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard (B. Councill and Heineman, 2001).

A component model defines hereby "specific interaction and composition standards" (B. Councill and Heineman, 2001). WebComposition/EUD components behave as fully-fledged applications with arbitrary business logic. Depending on the application domain, they can provide access to local or remote data sources, perform business-related calculations, enable telecommunication etc. (cf. *Domain Orientation* guideline). The intent behind taking fully-fledged applications as elementary building blocks is to lower the learning curve of the approach (cf. Live Programming principle). Non-programmers should be able to reuse their experience in usage of traditional Web and smartphone applications. WebComposition/EUD components expose their functionalities over two interfaces: a GUI and a so-called Inter-Component Communication Interface. The GUI is primarily used for interactions with human users and enables *visual* access to (eventually parts of) component logic and data. The Inter-Component Communication Interface (ICCI) targets software entities external to the component and enables *programmatic*

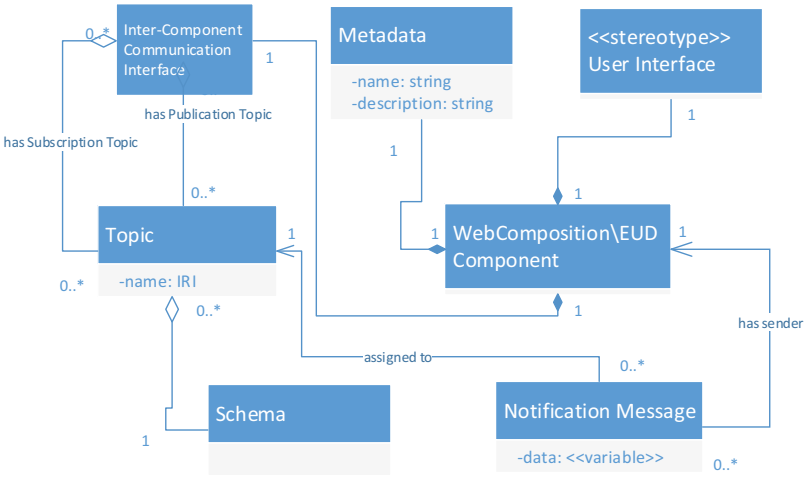


Figure 4.2.: WebComposition/EUD Component Model

access to its (eventually different parts of) logic and data. The ICCI reflects topic-based publish-subscribe capabilities of components. The publish-subscribe-based communication model has been chosen due to the low degree of coupling it induces (in space, time and synchronization) (Eugster et al., 2003). The UI and ICCI interfaces enable UI composition of components, i.e., aggregation on one canvas and definition of state synchronization logic (Wilson et al., 2012).

The conceptual model of WebComposition/EUD components is illustrated in Figure 4.2.

Definition 4.3.2. A WebComposition/EUD component c is a tuple $\langle M, GUI, ICCI \rangle$ with

- $M = \langle name, description \rangle$ being the basic component *Metadata*, which consists of at least the name and the description of the component. The two describe content and purpose of a component from end users' point of view. They are mainly used for component discovery during composition development.

- GUI being the *User Interface* of the component represented by a Document Object Model (DOM) tree (Le Hors et al., 2004). Nodes of the tree are HTML elements, which are characterized by a name, content and a set of attributes. HTML elements are associated with element-specific events that can be triggered either upon interaction with user or using DOM API.

- $ICCI = \langle T, PUB, SUB \rangle$ being the *Inter-Component Communication Interface* with:

  - $T = \{t_n = \langle name, schema \rangle\}$ being the *Topic* vocabulary supported by the component. Each topic t_n defines semantics of messages that can be assigned to t_n during inter-component communication (cf. Definition 4.3.3). $name$ is an Internationalized Resource Identifier (IRI) and should uniquely identify a topic in some context. $schema$ defines data type of messages that can be sent or received over the topic.

  - $PUB = \{p_i | p_i \in T\}$ being the set of *Publication Topics* supported by the component c. A publication is an emission of a message as a result of component-internal events such as interactions with user or updates of underlying data sources. Outgoing messages (cf. Definition 4.3.3) are assigned to publication topics p_i and contain data relevant to the occurred event.

  - $SUB = \{s_i | s_i \in T\}$ being the set of *Subscription Topics* supported by a component. A subscription is an assurance to receive and process incoming messages assigned to specific topics (cf. Definition 4.3.3). The messages are handled by component-internal routines that are defined by the component itself.

Messages that can be issued or received by components follow the model from Definition 4.3.3.

Definition 4.3.3. A *notification message* m is a tuple $\langle c, t, data \rangle$ with c being a WebComposition/EUD component(sender of the message), $t \in c.PUB$ being the topic, the message has been published on, and $data$ being the content of the message, which is valid according to the scheme $t.schema$.

An example of a WebComposition/EUD component is an application that shows emergency incidents around some given area. The model of the component can be defined as follows:

- $name =$ *"Emergency Incidents Map"*
- $description =$ *"Displays emergency incidents, locations of water level monitoring stations and Web cams around a given location"*
- GUI of the component draws an interactive map with dragging and zooming capabilities. Emergency incidents, Web cameras and water level monitoring stations are marked using dedicated markers. Clicks on emergency incident markers result in a pop-up window that displaying details of respective incident. Clicks on other markers make the component issue notification messages with Web camera or water level measurement station identifiers.
- $ICCI$ consists of one subscription on topic $t_{Location} = \langle http : //artifact\ library.example.org/topics/location, S_{Location} \rangle$, and two publications $t_{Station} = \langle urn : topic : station, S_{Station} \rangle$, $t_{Webcam} = \langle urn : topic : webcam, S_{Webcam} \rangle$. $t_{Location}$ describes messages that contain "longitude" and "latitude"of point to center the map around. $t_{Station}$ and t_{Webcam} define the only concept "id", which identifies water level measurements stations and Web cameras respectively. Schemata $S_{Location}$, $S_{Station}$, S_{Webcam} describe data types of messages that can be sent or received over respective topics. Example data types can be sets of value-pairs $\{\langle v, k \rangle\}$ with v and k being strings.

4.3.2 Composition Model

The WebComposition/EUD composition model (cf. Figure 4.3) foresees template-based aggregation and so-called customizable choreography of WebComposition/EUD components. The former enables selection and assignment of components to template placeholders. Placeholders should simplify and speed up the composition process by avoiding overlapping between components. Choreography refers to the fact that the composition model doesn't require explicit definition of communication channels between components if their ICCI interfaces are considered to be "compatible" (cf. Section 4.3.1). Instead, the channels are established automatically according to the topic-based publish-subscribe interaction scheme (Eugster et al., 2003). The motivation behind choosing customizable choreography as a communication model is to speed up and to simplify the composition process. In contrast to orchestration approaches, components establish communication channels automatically as soon as they are added to the composition. The customization aspect enables their manual modification to meet specific user requirements. Customization of communication channels takes place either by disabling communication over some topic, by isolating components or by defining topic transformations. The complexity of development process increases from *aggregation of visual components* over *communication restriction* to *topic transformations*. The proposed model can be considered as an extension of the basic UI mashup model introduced in (Wilson et al., 2012).

Definition 4.3.4. A WebComposition/EUD composition cc is a tuple $\langle M, C, VP, CP, ICC \rangle$ with

- $M = \langle name, description \rangle$ being the basic composition *Metadata*, which consists of at least the name and the description of the application. The two describe its content and purpose from end users' point of view.
- $C = \{c_i\}$ being the set of WebComposition/EUD components as defined in Section 4.3.1

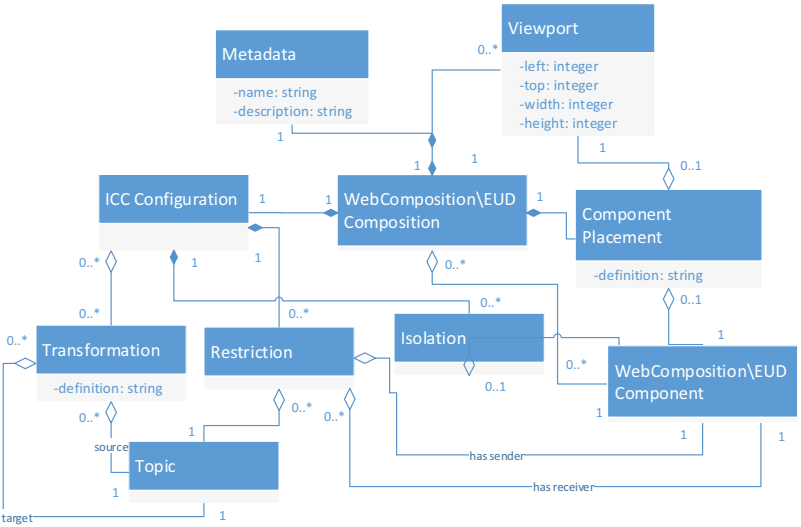


Figure 4.3.: WebComposition/EUD Composition Model

- $VP = \{v_j = \langle left, top, width, height \rangle\}$ being the set of *Viewports* for rendering of components
- $CP = \{(v_j, c_i) : v_j \in VP, c_i \in C\}$ being the *Component Placement*
- $ICC = \langle R, I, F \rangle$ being the Inter-Component Communication Configuration with
  - $R = \{r_l : r_l = \langle c_{l,sender}, c_{l,receiver}, t_l \rangle\}$ being *Restrictions* on communication between components $c_{l,sender}, c_{l,receiver} \in C$ over some arbitrary topic t_j
  - $I \subseteq C$ being the set of *Isolations*, i.e., components that are forbidden to communicate with others.

- $F = \{f_n : t_g \longrightarrow t_h\}$ being the set of *Transformations*, which are functions that map content of messages published on topic t_g into schema-compliant content for topic t_h.

The communication rules between components are defined as follows:

- Components c_{sender} are not allowed to receive or send any messages if $c_{sender} \in I$
- Components c_{sender} are allowed to emit (publish) messages on topics t if $t \in c_{sender}.PUB$. As described in Section 4.3.1 components decide autonomously, i.e., corresponding to their internal logic, when and which content to publish.
- Let $a = \langle c_{sender}, t, data \rangle$ be a notification message published by c_{sender}. The message a is delivered to all components $c_i \in C : c_i \neq c_{sender}$ if and only if
  - $t \in c_i.SUB$ and
  - $c_i \notin I$ and
  - $\nexists r_j \in R : r_j = \langle c_{sender}, c_i, t \rangle$.
- Let $a = \langle c_{sender}, t, data \rangle$ be a notification message published by c_{sender} and $\exists f \in F : t \longrightarrow t^* \in \bigcup_{\forall c \in C} c.SUB$. The message $a^* = \langle c_{sender}, t^*, f(a.data) \rangle$ is delivered to all components $c_i \in C : c_i \neq c_{sender}$ if and only if
  - $t^* \in c_i.SUB$ and
  - $c_i \notin I$ and
  - $\nexists r_j \in R : r_j = \langle c_{sender}, c_i, t^* \rangle$.

Communication between components builds up on the topic-based publish-subscribe messaging style (Eugster et al., 2003). Components support it by design and, thus, initiate mutual communication without any explicit configuration in the composition model (providing interface compatibility). This principle simplifies and accelerates composition process as no explicit configuration should be specified in order to let components communicate (Chudnovskyy, Nestler, et al., 2012; Isaksson and Palmer, 2010). To avoid rigidity of the composition and enable its extensibility (Tschudnowsky, Pietschmann, Niederhausen, Hertel, et al., 2014) the default topic-based publish-subscribe behavior can be customized. The customization takes place using isolations, restrictions and topic transformations. A component can be isolated either by defining restrictions for all of its in- and outgoing connections or by adding it to the set I. The difference is that the first configuration isolates the component from its current "neighbors", whereas the latter one disables its communication also with new components if added. The transformation function can be used to enforce communication between two components that have incompatible ICCIs. The function creates new notification messages out of actually published ones. The new messages are delivered to all components subscribed to the specified topic.

An example of a composite application could be an emergency response DSS described in Section 2.1.1. It could consist of the following WebComposition/EUD components: a map (cf. Section 4.3.1), two components visualizing information provided by a specified water level measurement station, one component displaying view of a specified Web camera, and one SMS component (cf. Figure 4.4).

The goal of the DSS can be to provide aggregated information on emergency incidents during natural disasters such as in case of flood. The composite application enables comparison of flood levels in different locations, access to live views on the most critical places and sharing of information with other people. The composition model $CC_{EmergencyResponse}$ of the application looks like as following:

- $name =$ *"Emergency Response"*

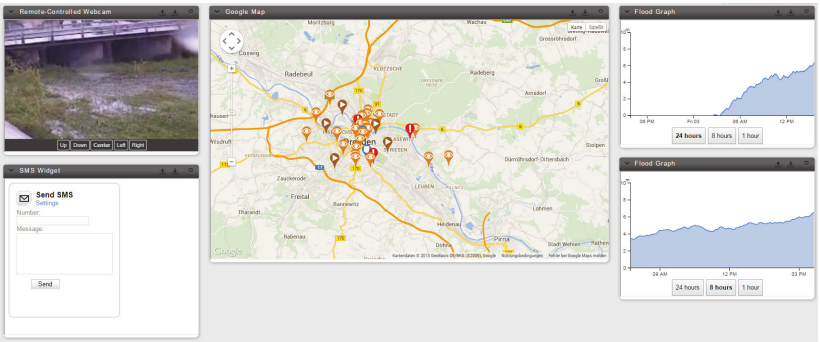


Figure 4.4.: Example DSS for Flood Catastrophe Management

- $description =$ *"The application provides aggregated information on emergency incidents occurred as a result of flood catastrophe"*
- $C = \{c_{Map}, c_{Levels,1}, c_{Levels,2}, c_{Webcam}, c_{SMS}\}$
- $VP = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ being available viewports for rendering components
- $VPA = \{(v_1, c_{Levels,1}), (v_2, c_{Levels,2}), (v_3, c_{Map}), (v_5, c_{SMS}), (v_6, c_{Webcam})\}$ being the viewport-component associations
- $ICC = \langle R, I, F \rangle$ being the Inter-Component Communication (ICC) configuration with
  - $I = \{c_{Webcam}\}$ being the set of isolations that makes Web camera widget keep its view, even if a marker in c_{Map} gets activated (e.g., to avoid accidental shifts).
  - $R = \{\langle c_{Map}, c_{Levels,1}, http://artifactlibrary.example.org/topics/station \rangle\}$ being the set of restrictions that forbid communication between c_{Map} and $c_{Levels,1}$.

– $F = \{f\}$ being the set of transformation functions with the only function f that maps publication $http://artifactlibrary.example.org/topics/water-level$ of $c_{Levels,1}$ on subscription topic $http://artifactlibrary.example.org/topics/sms$ of c_{SMS}. The mapping algorithm can be a concatenation of all elements from original water level data assigned to the element *text* of target notification message.

Users can interact with each aggregated component – they can move and explore the map, modify Web camera and water levels display parameters or send SMS. Due to the default communication behavior of the components and their compatible interfaces, both water level components will always display the data related ti the marker selected on the map. This is, however, undesired, if two measurements should be compared. The specified restriction prevent $c_{Levels,1}$ from reaction on marker selection event and "freezes" its view. $c_{Levels,2}$, however, will continue to receive messages from c_{Map} and will adapt its view correspondingly. The same effect is achieved by isolating component completely, cf. the isolation set I. The difference is that $c_{Levels,1}$ is allowed to receive or send messages to other components, while c_{Webcam} not. Finally, in case, the SMS component should be automatically populated with measurement data from $c_{Levels,1}$, a new topic transformation function f is defined that performs content mapping between publications and subscriptions of the two components.

4.4 Process Model

According to the principle of *Separation of Responsibilities* the WebComposition/EUD process model distinguishes among several logical roles involved into development of WebComposition/EUD compositions (cf. Figure 4.5).

Component Developers (CD) create WebComposition/EUD components that address business-related tasks and that can be com-

bined together to more complex solutions. Component Developers are skilled programmers with domain expertise.

Component Communication Experts (CCE) ensure composability of components by adjusting their ICCIs or by implementing ICCI transformations. Although ICCIs can be implemented by Component Developers, Component Communication Experts resolve incompatibility issues that might arise, if ICCIs utilize different data structures or data types. Average programming skills and some domain expertise are required to perform this task.

Artifact Library Managers (ALM) maintain public or organization-internal library of WebComposition/EUD components and their compositions. To enable efficient discovery and reuse, library artifacts are annotated. Relationships and usage possibilities are made explicit. For maintenance activities Artifact Library Managers require strong domain expertise as well as understanding of component behavior and implementation.

Composition Developers (CSD) create and use DSSs by combining WebComposition/EUD components into one composite application. The resulting solutions are used to analyze data or access functionalities required for decision making. Composition Developers are the main target group of this thesis. We assume that Composition Developers do not have any programming skills but are experts in their problem domain. However, with growing expertise Composition Developers can customize their solutions more flexibly and share their compositions with others.

Based on the skills of stakeholders introduced in Section 2.2 the role of Composition Developers in the WebComposition/EUD process is assigned to Decision Makers. Software Providers should play the remaining roles.

To address the requirements elicited in Chapter 2, the WebComposition/EUD process model builds up on several existing state-of-the-art solutions. The core of the approach is the WebComposition Process

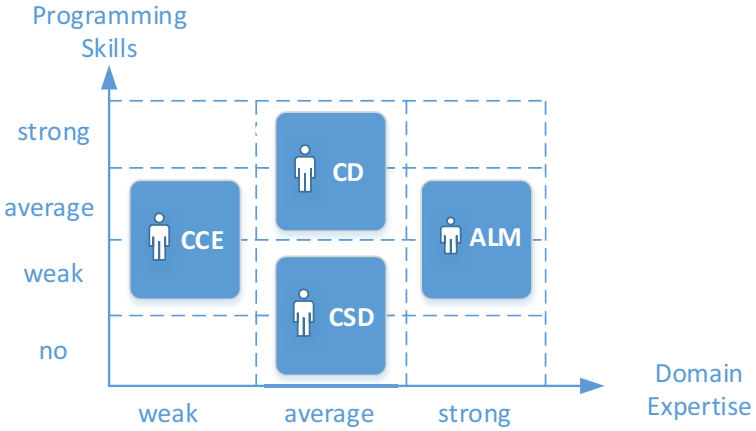


Figure 4.5.: WebComposition/EUD Role Model

Model (Gaedke and Gräf, 2001). The motivation of building upon this model is its strong component-orientation, extensibility and openness. Furthermore, it explicitly addresses continuous evolution of software, which is also one of the objectives of the thesis. The proposed extension of the WebComposition Process Model adds steps, artifacts and roles that are specific to end-user development under time-pressure.

The WebComposition/EUD process model distinguishes between two coarse-grained phases: a so-called *Bootstrap Phase* and a *Continuous Evolution Phase* (cf. Figure 4.6). The former has a goal to establish an initial knowledge base and a set of general-purpose components for later DSS development. The latter (derived from the WebComposition Process Model) describes the process of iterative DSS evolution performed by end users.

The Bootstrap Phase starts with so-called *Domain Exploration*. The step has a goal to identify common data sources and functionalities that can support daily activities of decision makers from a particular domain. Based on recognized usage patterns Artifact Library Managers define groups of coarse-grained entities that should be later wrapped

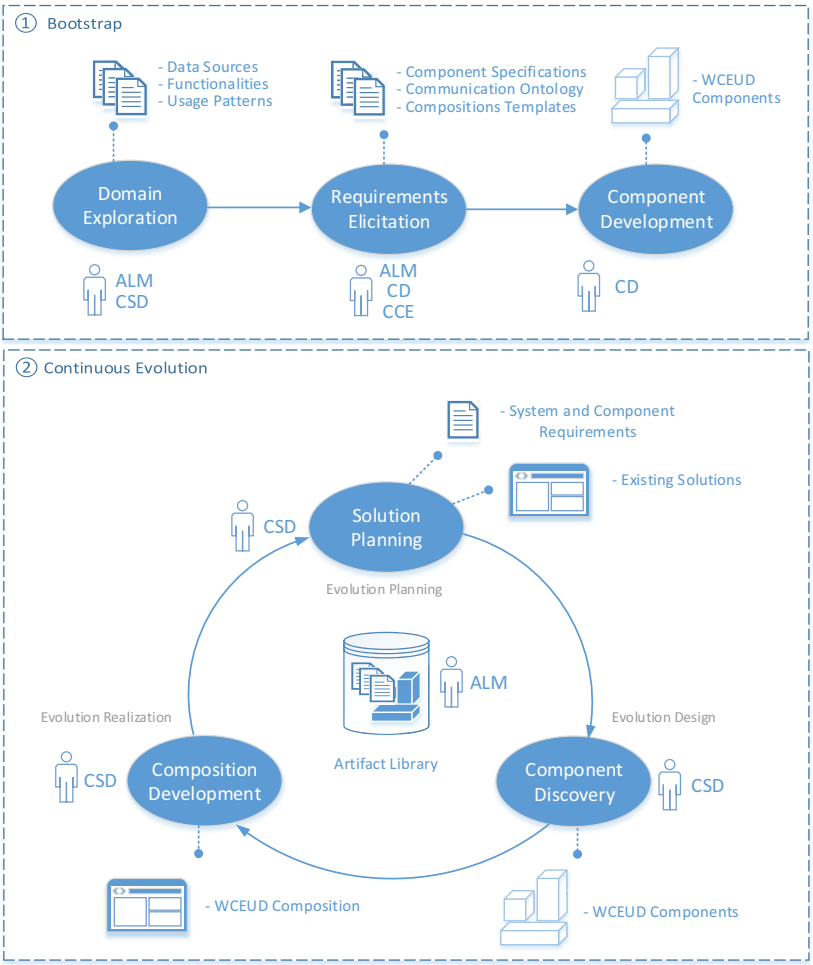


Figure 4.6.: WebComposition/EUD Process Model

into reusable WebComposition/EUD components. Together with Component Developers and Component Communication Experts functional and non-functional requirements on the future components are elicited (*Requirements Elicitation* step). Component Communication Experts define a common communication ontology and derive ICCI interfaces for seamless composability of components. Component metadata and capabilities are documented. Artifact Library Managers also capture relationships between components, potential component compositions and business goals addressed by these compositions. This initial knowledge base builds the basis for later DSS development by Composition Developers. Once the requirements are elicited Component Developers start designing, implementing and testing envisioned pieces of software (*Component Development* step). Development process to be used is not restricted and can be chosen by respective development teams freely. After components are developed Artifact Library Managers make them available for composition and, thus, finish the bootstrap phase.

The Continuous Evolution Phase promotes iterative development of DSSs based on reuse of existing components and their compositions. Similarly to the original WebComposition Process Model three iterative steps around a centralized reuse repository are defined. The first one called *Solution Planning* corresponds to the original Evolution Analysis step. Composition Developers analyze, which functionalities are required to solve their current problem and identify existing solutions that fit their envisioned DSS best. For this purpose they either browse the Artifact Library maintained by Artifact Library Managers manually or make use of assisting discovery mechanisms. If no existing solution is available that would fit the business goals, development starts with an empty solution. Otherwise, the discovered solution is reused and a copy of its composition model is created. The second step called *Component Discovery* corresponds to Evolution Design in the original model. Components that provide required functionality, are identified. The Artifact Library acts as a source of potential candidates. If no appropriate component is found or modifications to existing ones are needed, a request to Artifact Library Manager can be issued. The latter eventually initiates component development or modification as it is done in the Bootstrap Phase. During the last *Composition Development*

step, which corresponds to Evolution Realization step in the original model, discovered components are integrated into existing solution. Properties, layout and communication capabilities of components in the solution are configured. In the sense of Live Programming principle these activities overlap with the actual usage of the solution, so that each configuration change can be immediately tested and evaluated by Composition Developers. Stable and reusable solutions are reviewed and published by Artifact Library Managers to be reused as templates for future compositions.

Similarly to the original process model the presented steps can be performed in cycle. By enriching existing solutions with new components changing requirements and new use cases can be supported.

4.5 Methods

The WebComposition/EUD approach introduces three techniques that enable systematic execution of the most complex and time-consuming activities in the development process. The addressed activities are 1) development of WebComposition/EUD components, 2) development of component-based DSSs and 3) management of various WebComposition/EUD artifacts.

The first one called **Component Conversion and Enrichment** enables time- and cost-efficient production of model-compliant components out of legacy ones. Instead of developing required functionality from scratch, the method proposes to reuse existing code of Web-based widgets with adaptations that can be described by a set of rules specific to the source widget format. To equip the resulting component with inter-component communication capabilities a UI-based configuration technique is proposed. To speed up the development process it enables interactive demonstration of the desired behavior instead of direct source code adaptations. Details of the Component Conversion and Enrichment method are described in the context of implementing tools (cf. Section 7.2 and Section 7.3).

The second technique called **Dialog-based Discovery** aims at shortening the time required for Component Discovery and Composition Development steps. It proposes to use interactive dialog-based UI for expressing business goals of decision makers. Instead of discovery, configuration and assembly of components from scratch, it promotes customization of existing solutions that at least partially satisfy the stated goals. The dialog-based interface should minimize potential errors in solution configuration and decrease the complexity of the system. The reuse aims at increasing quality of the resulting product and minimizing the development time. Section 6.2 describes a software tool that implements the Dialog-based Discovery technique.

The last technique called **Linked Artifact Metadata** has a goal to simplify discovery and access to artifacts produced during the development process. It foresees publication and linking of metadata of artifacts using Linked Data principles (Berners-Lee, 2009). The technique promotes and simplifies internal and external reuse of WebComposition/EUD components and compositions. Component Developers can find and access component description in a uniform way. Explicit linking and use of shared vocabularies should enable Composition Developers to discover new solutions or components both within and outside one organization. Section 7.4 describes a software tool that supports and partially automates application of the Linked Artifact Metadata technique.

4.6 Tools

The WebComposition/EUD tools support various activities and roles within the introduced process model (cf. Figure 4.7). In line with the thesis objectives the tools aim at 1) enabling development of DSSs by end-users, 2) speeding up time-consuming activities and 3) ensuring time- and cost-efficient evolution of produced artifacts. The objectives enable grouping of the proposed tools into three corresponding categories: the composition platform, development assistance and evolution assistance mechanisms, which are introduced next.

4.6.1 Composition Platform

The Composition Platform acts as a run-time environment and a visual editor for WebComposition/EUD composition models (cf. Section 4.3.2). The platform addresses Composition Developers as its main user group and enables interactive customization and execution of DSSs. Integrated component browser provides access to available WebComposition/EUD components, which can be instantiated and aggregated on one canvas. The composition platform blurs the edge between development and usage activities (cf. Live Programming principle), i.e., any change to the composition model becomes immediately visible and explorable to platform users. The Gentle Slope of Complexity principle is applied while offering different types of customization of component communication behavior. The simplest activity is the isolation of components from mutual communication. For definition of restrictions a basic understanding of underlying communication model is required. Finally, the most challenging task is specification of mappings between various communication channels.

4.6.2 Development Assistance

Various automation mechanisms aim at simplifying and accelerating development of DSSs by Composition Developers.

Automatic Discovery and Composition Engine The first tool called Automatic Discovery and Composition Engine (ADCE) (cf. Section 6.2) supports novice Composition Developers in the first steps of DSS development and suggests possible solutions for common business goals. The tool applies the Dialog-based Discovery technique (cf. Section 4.5) and acts as an expert system for various problem domains. Composition Developers start with selection of business goals from a knowledge base and refine them during a question-answering game. If no ready-to-use solution for user problem exists, ADCE provides a possibility to describe required data sources and functionalities that will be automatically searched for and eventually included into the first solution prototype.

After automatic instantiation the prototype can be customized or extended using facilities provided by the composition platform.

Double Input Detector The second tool called Double Input Detector (DID) (cf. Section 6.4) simplifies usage of DSSs in time-pressuring situations and automates various manual actions such as repeated user inputs or form submissions. For this purpose it observes user interactions with GUIs of aggregated components and creates logical connections between GUI elements considered to be related. The "related" relationship is computed dynamically by comparing user inputs into the two elements. Once a logical connection is established, the tool automatically synchronizes content of the elements without the need of time-consuming and potentially error-prone manual input. As a result users can access data and functionality offered by different components faster and more reliably.

Loop Detection Facilities The third tool called Loop Detection Facilities (LDF) (cf. Section 6.3) is capable of detecting erroneous configurations in the composition model and preventing faulty behavior of the resulting solutions. The goal is to increase the reliability and fault-tolerance of the platform. LDF consists of two mechanisms that can detect and prevent potentially unwanted communication between components. The first one analyzes composition model and ICCIs of aggregated components and notifies users, if intense mutual communication among subset of components is possible. The second mechanism analyzes actual communication among components at run-time and produces warnings, if it is considered to be potentially long-lasting and *harmful* to the application. LDF enables timely feedback on eventual errors in the application and can prevent them by automatically adjusting the composition model.

4.6.3 Evolution Assistance

Evolution of existing DSSs is supported by a toolkit consisting of three mechanisms.

WebComposition/EUD Component Converter The first tool called *WebComposition/EUD Component Converter (WebComposition/EUD-CC)* enables efficient development of WebComposition/EUD components out of components of other types and models (cf. Section 7.2). It automates the first step of the Component Conversion and Enrichment technique (cf. Section 4.5). The goal is to efficiently enrich the set of data sources and functionalities available for integration and, thus, to enable adaptation of existing DSS solutions towards changing requirements. The target group of the tool are Component Developers. WebComposition/EUD-CC automatically maps structure and metadata of components of other types on corresponding elements within WebComposition/EUD components. To simplify and automate conversion of many components a configurable batch mode is implemented.

WebComposition/EUD ICCI Extender The second tool called *WebComposition/EUD ICCI Extender (WebComposition/EUD-IE)* enables efficient extension of ICCI of WebComposition/EUD components (cf. Section 7.3). It automates the second step of the Component Conversion and Enrichment technique (cf. Section 4.5). The goal of WebComposition/EUD-IE is to accelerate the extension process and to avoid manual error-prone source code adaptations. The target group is Component Communication Experts. WebComposition/EUD-IE offers a visual environment that allows interactive specification of new publications and subscriptions. The definition takes place in the PBD fashion (Lieberman, 2001): one performs a sequence of actions on the GUI of a component and the tool automatically converts it (or its subset) into new ICCI primitives. The latter represent high-level functionalities that encapsulate low-level GUI events and operations. New ICCI primitives are tightly coupled to the GUI of the original component. They propagate changes of interaction elements and enable indirect invocation of their operations.

WebComposition/EUD Artifact Library The last facility from the evolution support toolkit is the so-called WebComposition/EUD Artifact Library (cf. Section 7.4). The library enables efficient management of various types of reusable artifacts involved into development of DSSs. The target group of the tool are Artifact Library Managers. The

library stores available WebComposition/EUD components, their metadata, shared composition models and various ontology definitions. The resources can be accessed and managed over a configurable RESTful interface. A built-in authentication and authorization mechanism enables fine-grained access control to library resources.

4.7 Summary

This chapter introduced the WebComposition/EUD approach for time-constrained development of DSSs by end users. The basic idea of the approach is to enable end-user-friendly development and customization of DSSs based on compositions of specialized components. The approach consists of several measures to ensure quality of the development process and of the produced artifacts. Its core principles promote artifact reuse, gentle slope of complexity and separation of responsibilities. The process model based on the WebComposition approach describes roles, phases and activities for systematic DSS development by end users. The formalisms include component and composition models, algorithms and reference architectures that should hide complexity of underlying technology behind appropriate abstractions. The methods define several techniques for time- and cost-efficient development and evolution activities. Tools, which include the WebComposition/EUD composition platform and various assistance mechanisms, provide implementations of the proposed ideas and techniques. The next three chapters provide a detailed view on the tools and methods of the WebComposition/EUD approach.

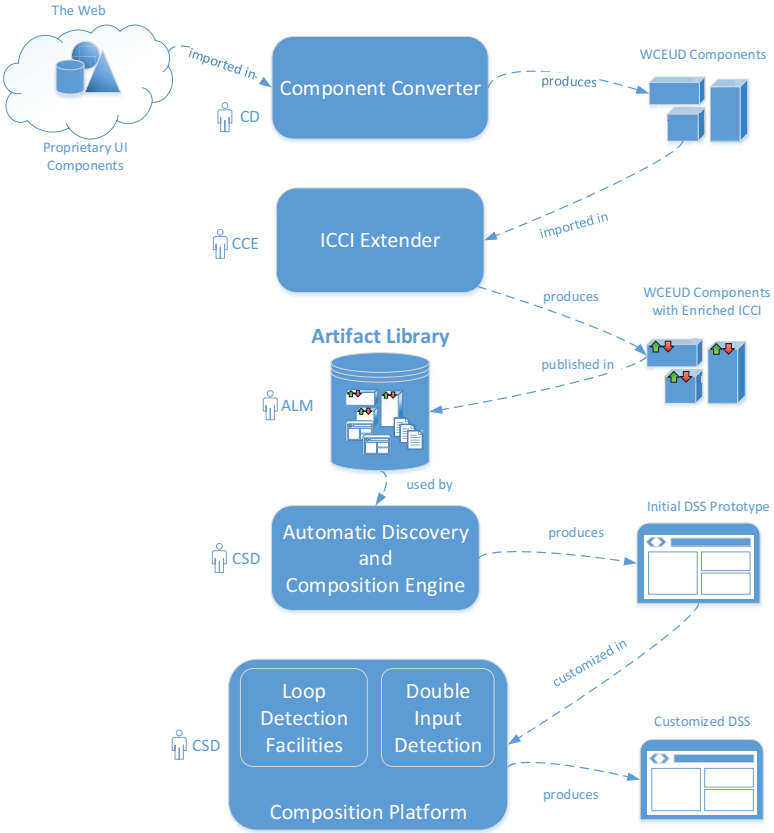


Figure 4.7.: WebComposition/EUD Toolkit and Corresponding Target Roles

5 Composition Platform

This chapter introduces the WebComposition/EUD composition platform, which is the core of the whole technological toolkit. First, appropriate requirements on design and implementation of the platform are defined. Then a conceptual architecture of the platform is presented and its implementation using Web technologies is shown. Finally, results on usability evaluation of the platform are given.

5.1 Research Questions

Development environments are collections of tools and workbenches to support professional software development process (Fuggetta, 1993). WebComposition/EUD approach, however, targets end users and aims at simplifying the process. The main challenge addressed by this chapter is how to enable access to and use of WebComposition/EUD framework. The research questions addressed are:

Research Question 1 Is it possible to define a technology-independent platform architecture for storage, instantiation and manipulation of WebComposition/EUD models? This question aims at identifying abstract components, relationships, data and control flows that can be used as a footprint to build technology-specific solutions.

Research Question 2 Do the proposed awareness and control facilities make users more efficient in simple data comparison tasks? This questions addresses the usability of the proposed concepts and evaluates if they decrease time required for solving given data comparison tasks.

Research Question 3 Does the proposed Transformation Editor match skills of Composition Developers? This question analyzes confidence of users while applying the tool and evaluates its usability.

5.2 Requirements

The main goal of the WebComposition/EUD composition platform is to enable development of DSSs by iterative customization of discovered solutions. To achieve this goal the platform should fulfill the following requirements (implied by the ones from Chapter 2 and principles from Section 4.2):

Customization Support As discussed in Section 4.4 the customization step foresees five main activities:

1. Modifying the metadata of the solution.
2. Modifying the set of components that constitute the DSS.
3. Modifying the set of available viewports
4. Modifying component-viewport assignments
5. Modifying the communication configuration (isolating components, defining restrictions and transformation functions)

The platform should enable all the above activities.

Ubiquity Time-pressuring situations require DSSs that are available on any device at any time. The composition platform should, therefore, produce solutions that do not require any specialized hard- or software to run.

Usability The main target group of the composition platform are decision makers without programming skills. Therefore, the composition environment should be easy to learn, easy to apply and efficiently support solution of domain problems.

5.3 Conceptual Architecture

The conceptual architecture of the WebComposition/EUD composition platform is presented in Figure 5.1. It represents a slight modification of the OMELETTE mashup architecture (OMELETTE Consortium, 2013a), which has been developed by consortium of the EU FP7 Research Project OMELETTE[1] with participation of the author. Similarly to the original proposal the WebComposition/EUD architecture aims at establishing a platform-independent infrastructure for end-user mashup development. The introduced modifications address manipulation of WebComposition/EUD specific models (restrictions, isolations and topic transformations), foresee integration of future end-user assistance mechanisms (by means of Extension modules) and abstract away from some scenario-specific components (Service Mashup Environment, OMELETTE Information Store etc.).

The main component of the architecture is the *Live Composition Editor*, which acts as a development and run-time environment for WebComposition/EUD components and compositions. Customization activities are performed directly on running applications by means of direct component manipulation, additive views and context menus. The editor updates the underlying Composition Model on every change and immediately reflects the changes back in the running instance (cf. Live Programming principle, Section 4.2). Composition models can be ex-

[1] `http://cordis.europa.eu/project/rcn/95584_en.html`, Retrieved: 6.8.2016

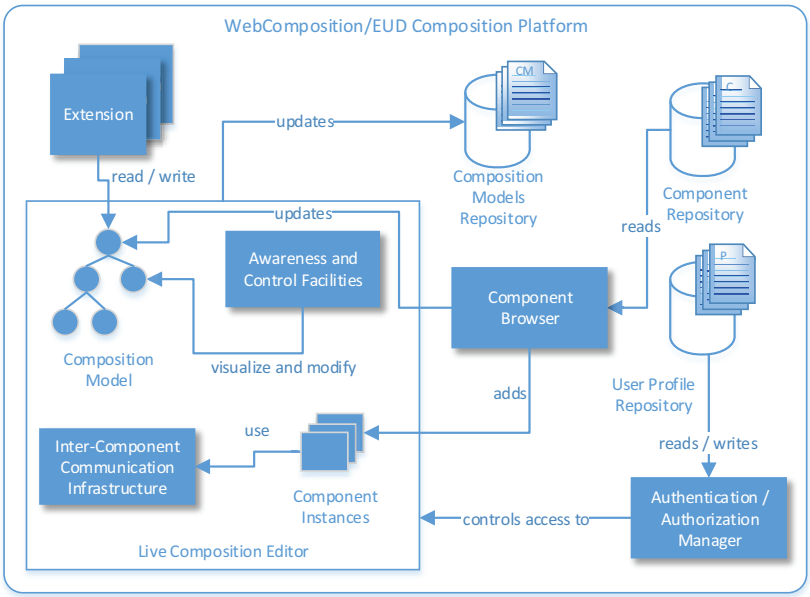


Figure 5.1.: Conceptual Architecture of the WebComposition/EUD Composition Platform

ported into the *Composition Model Repository*. Editor interface enables integration of additional model observation and modification modules (*Extensions*) such as ADCE (cf. Section 6.2) or LDF (cf. Section 6.3). Component packages are stored in the *Components Repository*, which can be explored with the help of the *Component Browser*. Run-time communication between components is enabled by a dedicated *ICC Infrastructure*, which distributes component notification messages according to the communication rules defined in Section 4.3.2. Finally, *Authentication and Authorization Manager* enables authentication, authorization and management of platform users, whose profiles are stored in the *Profile Repository*.

5.4 Implementation

The composition platform has been implemented using standard Web technologies. The choice of the Web as a technology basis is motivated by device and platform independence of resulting solutions as well as potential for installation and maintenance cost reduction especially in case of Cloud deployment (Miller, 2008). Furthermore, popularity of the Web and its wide acceptance resulted in concepts and interaction patterns that became well-understood and widely applied by users even without programming skills. In the following, Web realization of WebComposition/EUD components, composition models and the above architecture is presented.

5.4.1 WebComposition/EUD Components

WebComposition/EUD components are implemented on the basis of W3C Packaged Web Apps (Widgets) specification (Cáceres, 2012). The choice of this particular technology is motivated by its vendor-independence, openness and open-source tool support. Similarly to the Packaged Web Apps WebComposition/EUD components are "full-fledged client-side applications that are authored using technologies such as HTML and then packaged for distribution" (Cáceres, 2012). An app



Figure 5.2.: Example of a W3C Packaged App Content (Source: Cáceres, 2012)

package is a ZIP-archive with file extension **.wgt*, which contains source code files and a configuration document. The source code consists of HTML, Cascading Style Sheets (CSS) and JavaScript files and their optionally localized copies. The configuration document describes app metadata (title, description, author, license etc.), requirements on the execution environment (proxy, communication libraries etc.) and default app preferences. An example of a W3C Web app package is depicted in Figure 5.2. The package is installed and executed in dedicated run-time environments called Widget Containers. The latter interpret the source code, provide required run-time services and implement JavaScript API defined by the specification. Apache Wookie[2] is an example of a Web-based widget container that enables hosting and integration of W3C Web apps into third-party applications.

WebComposition/EUD components are specialized W3C Web Apps that have been enriched with ICCI as proposed in Section 4.3.2. Each WebComposition/EUD component can produce and consume notification messages assigned to various logical channels, i.e., topics. Topics names are IRIs. In case of HTTP-IRIs they point to schemata of valid

[2] `http://wookie.apache.org`, Retrieved: 18.05.2015

topic messages. For simplicity and convenience purposes the resources behind the HTTP-IRIs are JSON-schema[3] documents.

The messaging API for WebComposition/EUD components is specified and implemented by OpenAjax Hub (OpenAjax Alliance, 2009). The OpenAjax Hub framework enables "isolation of components into secure sandboxes" and equips the isolated components with managed topic-based publish-subscribe infrastructure. For compliance with the WebComposition/EUD composition model the framework has been extended to support restrictions, isolations and transformation functions. Listing 5.1 exemplifies usage of OpenAjaxHub API for sending and receiving of notification messages.

The ICCI of WebComposition/EUD components is described in the package configuration document. Explicit declaration of ICC capabilities facilitates discovery and composition of components as well as analysis and control of ICC (cf. Figure 5.3).

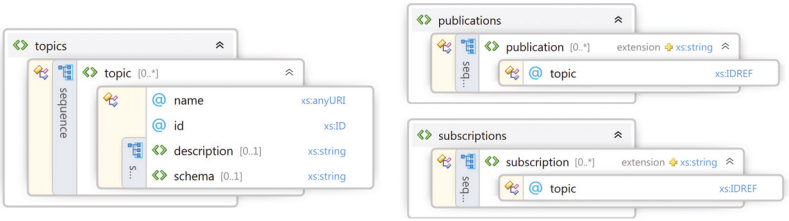


Figure 5.3.: Extensions of W3C Configuration Document to Describe ICC Behavior

The description uses new XML elements that describe topic vocabulary, publications and subscriptions supported by a component. The topic vocabulary consists of references or definitions of used topic IRIs, their textual descriptions and corresponding JSON schemes. Component publications and subscriptions reference elements from the declared topic vocabulary. Listing 5.2 shows an excerpt from the configuration document that describes the example map component from Section 4.3.1.

[3] `http://json-schema.org`, Retrieved: 22.05.2015

Listing 5.1: Inter-Component Communication using OpenAjax Hub

```
...
/* Subscribe for map center messages */
hub.subscribe(
  "http://artifactlibrary.example.org/topics/location",
      centerChanged);
...
function centerChanged(topic,data,meta) {
  pos = new google.maps.LatLng(data.latitude, data.
      longitude);
  map.panTo(pos);
}
...
/* Publish identifiers of measurement stations */
for(int i=0;i<stations.length;i++) {
  station = stations[i];
  marker = createGenericMarker(new google.maps.LatLng(
      station.latitude, station.longitude),"
      marker_station","Station:␣"+station.id);
  google.maps.event.addListener(marker, 'click',
    function(event)
    {
      ...
      hub.publish("http://artifactlibrary.example.org/
          topics/station",station);
      ...
    }
  );
}
```

It references one topic ($t_{Location}$), declares two other ($t_{Station}$ and t_{Webcam}) and defines two component publications and one subscription. The referenced topic definition is presented in Listing 5.3.

Listing 5.2: Extension of the W3C Configuration Document

```
<feature name="http://www.openajax.org/hub"
    xmlns:oa="http://www.openajax.org/hub">
  <!-- Declaration of supported vocabulary -->
  <oa:topics>
    <oa:topic id="tLocation"
        name="http://artifactlibrary.example.org/topics/location"
        />
    <oa:topic id="tStation" name="topic:station">
      <oa:description>Identifier of a water level measurement
          station</oa:description>
      <oa:schema>
      <![CDATA[
      {
        "type":"object",
        "properties":{
          "id":{"type":"string"}
        }
      }
      ]]>
      </oa:schema>
    </oa:topic>
    <oa:topic id="tWebcam" name="topic:webcam">
      <oa:description>Identifier of a Web camera</oa:description>
      <oa:schema>
      <![CDATA[
      {
        "type":"object",
        "properties":{
          "id":{"type":"string"}
        }
      }
      ]]>
      </oa:schema>
    </oa:topic>
  </oa:topics>
```

```
  <!-- Declaration of publications -->
  <oa:publications>
    <oa:publication topic="tStation"/>
    <oa:publication topic="tWebcam"/>
  </oa:publications>

  <!-- Declaration of subscribtions -->
  <oa:subscriptions>
    <oa:subscription topic="tLocation"/>
  </oa:subscriptions>
</feature>
```

Listing 5.3: Topic Declaration using a JSON-Scheme Document

```
curl http://artifactlibrary.example.org/topics/location

{
  "description":"GPS Coordinates",
  "type":"object",
  "properties":{
    "longitude":{"type":"number"},
    "latitude":{"type":"number"}
  }
}
```

5.4.2 WebComposition/EUD Composition

Compositions of WebComposition/EUD components are specified using configuration documents in the Open Mashup Description Language (OMDL) format (OMELETTE Consortium, 2013a). The OMDL specification enables interoperable description of metadata, structure and layout of widget-based mashups in different life-cycle phases. The specification has been reused due to its openness, similarity of concepts and availability of open-source tool support. OMDL described composite applications called *workspaces* and their building blocks called *apps*. The latter are instances of Web-based widgets that can be accessed through

an IRI. Various life-cycle phases of mashups (conceptual, logical and physical) are addressed by dedicated vocabularies and data types.

The composite application from Section 4.3.2 in the physical design phase can be described using OMDL as shown in Listing 5.4.

Listing 5.4: OMDL Description of the Emergency Response Application

```
<workspace xmlns="http://omdl.org/">
  <identifier>http://artifactslibrary.example.org/dss/379</identifier>
  <title>Emergency Response</title>
  <description>The application provides aggregated information on
      emergency incidents occurred as a result of flood
      catastrophe</description>
  <creator>Alexey Tschudnowsky</creator>
  <date>2013-06-18T14:39:58+0200</date>
  <layout>THREE COLUMNS</layout>

  <!-- Map -->
  <app id="a1">
    <link href="http://container.example.org/instance/12"
    type="application/widget" rel="source"/>
    <position>LEFT TOP</position>
  </app>

  <!-- Web Camera -->
  <app id="a2">
    <link href="http://container.example.org/instance/7"
    type="application/widget" rel="source"/>
    <position>LEFT MIDDLE</position>
  </app>

  <!-- Flood Levels 1 -->
  <app id="a3">
    <link href="http://container.example.org/instance/10"
    type="application/widget" rel="source"/>
    <position>RIGHT TOP</position>
  </app>

  <!-- Flood Levels 2 -->
  <app id="a4">
```

```
    <link href="http://container.example.org/instance/11"
    type="application/widget" rel="source"/>
    <position>RIGHT MIDDLE</position>
  </app>

  <!-- SMS Component -->
  <app id="a5">
    <link href="http://container.example.org/instance/4"
    type="application/widget" rel="source"/>
    <position>RIGHT TOP</position>
  </app>
</workspace>
```

To address the peculiarities of the WebComposition/EUD composition model, the OMDL format is extended with vocabulary to describe ICC configuration of a composite application (cf. Figure 5.4).

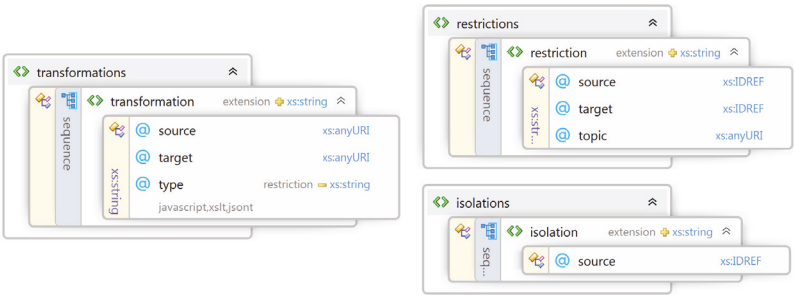


Figure 5.4.: OMDL Extensions to Describe ICC Configuration

The *restriction* element is used to specify component restrictions. The mandatory *source* attribute references a component, whose communication behavior should be restricted. The mandatory *target* attribute references the potential communication partner. If only *source* and *target* attributes are present, any communication between the referenced components is forbidden. The optional *topic* attribute defines restriction on a particular topic.

The *isolation* element with only attribute *source* references a component that should be completely excluded from ICC.

The *transformation* element defines topic transformation functions (cf. Section 4.3.2). The topics are referenced using mandatory attributes *source* and *target*. The algorithm type is given in the *type* attribute. The extension specifies three types: *javascript*, *jsont* and *xslt*. The algorithm itself is defined as opaque character data within the element. For the *javascript* type an implementation of a function *transform(topicSource,topicTarget,data)* should be provided that produces a new JSON object complying with schema of *topicTarget*. ICC configuration of the example application from Section 4.3.2 is given in Listing 5.5.

5.4.3 Run-Time Environment

The conceptual architecture presented in Section 5.3 is implemented on top of several open-source projects (cf. Figure 5.5).

The first one, *Apache Wookie*, is a Web-based container for W3C widgets. It acts as a run-time environment and enables hosting and integration of widgets into other Web applications. Furthermore, it provides instance management, proxy and instance-to-instance communication services. The second project, *Apache Rave*[4], is a Web-based platform for composition of mashups out of W3C and OpenSocial (OpenSocial and Gadgets Specification Group, 2014) widgets. It provides a visual editor and a run-time environment for composite applications. The editor enables interactive layouting of widgets in the boundaries of grid-based templates. Widgets are integrated via *iframes* and communicate using facilities of *OpenAjax Hub*. Apache Rave implements a browser for local and remote widget repositories.

Apache Rave and Wookie were extended to support models and processes specific to WebComposition/EUD framework. Structure of data-

[4] `http://rave.apache.org`, Retrieved: 20.05.2015

Listing 5.5: Example of Proposed OMDL Extension to Describe ICC Configuration

```
<!-- Isolated Web Cam component -->
<oa:isolations xmlns:oa="http://www.openajax.org/hub">
  <oa:isolation source="a2" />
</oa:isolations>

<!-- Forbid Communication between Map and One Flood Levels
     Component -->
<oa:restrictions xmlns:oa="http://www.openajax.org/hub">
  <oa:restriction
    source="a1"
    target="a3"
    topic="http://artifactlibrary.example.org/topics/gps"/>
</oa:restrictions>

<!-- Transform Flood Level Data into SMS Messages -->
<oa:transformations xmlns:oa="http://www.openajax.org/hub">
  <oa:transformation
    source="http://artifactlibrary.example.org/topics/floodlevel"
    target="http://artifactlibrary.example.org/topics/sms"
    type="javascript">
    <![CDATA[
      function transform(topicSource,topicTarget,data) {
                    output = {};
                    output.text = 'Current flood level in ' +
                              data.longitude + ',' + data.latitude + ' is ' +
                                    data.level + 'm'.
                    return output;
      }
      ]]>
  </oa:transformation>
</oa:transformations>
```

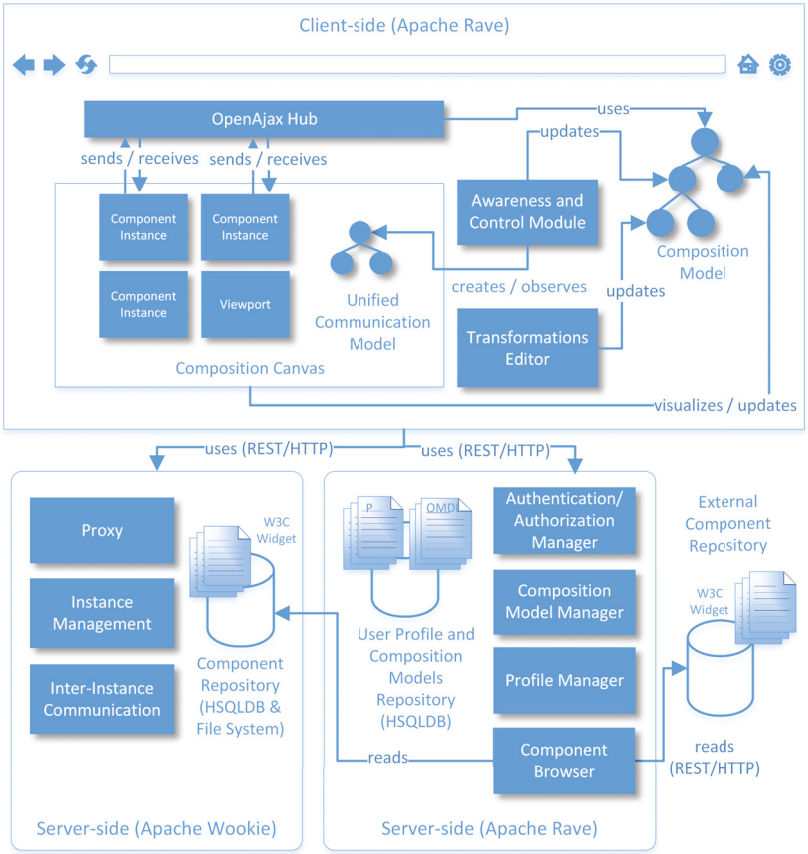


Figure 5.5.: Implementation of the Composition Platform

bases, APIs and deployment routines of the two applications have been adjusted to store ICC configuration of WebComposition/EUD components and compositions. OpenAjax Hub has been extended with control mechanisms that respect the ICC configuration. Finally, visual editor of Apache Rave has been enriched with *Awareness and Control* facilities, which visualize and enable modification of ICC configuration for Composition Developers.

5.4.4 Live Composition Editor

The implementation of Live Composition Editor is based on the interactive drag-and-drop editor provided by Apache Rave. Basic operations on the WebComposition/EUD composition model (configuration of metadata, layout and aggregated components) are built-in in Apache Rave. Configuration of the ICC behavior, however, is missing, so that new UI has been developed.

The implementation incorporates findings from the user study in (OMELETTE Consortium, 2013c) on implicit communication mechanisms. Implicitly established communication paths tend to impose usability problems that hinder acceptance of corresponding solutions. First, Composition Developers have to interact with components in order to learn data and control flows. This is not only a time-consuming activity, but can also affect live data. Second, direct and transitive connections are not obvious. The latter occur when one component triggers action in another one that in turn triggers a third component. Misinterpretation of relationships leads to unexpected behavior once components are removed or added. Finally, users typically do not see the data being transferred between components. Instead, they only perceive the effects of their transfer i.e., a receiving component is updated with new data. Missing understanding of data being exchanged leads to wrong expectations on component capabilities.

The implemented Awareness and Control facilities builds upon a dedicated model that describes communication relationships of components using a directed graph. The motivation for definition of a dedicated com-

munication model is that – from an end user point of view – components communicate in pairs by means of unidirectional message transfers. In terms of a concrete messaging pattern, messages may have different semantics e.g., invocation of a remote procedure, read/write operation to a shared memory or publication/subscription to some topic. Regardless of the pattern, the user-perceived result is that one component receives data from another one. These considerations build a basis for the communication model described below.

The WebComposition/EUD communication model is a graph $G = (V, E)$ with

- $V = \{v|v = (id, s)\}$ set of vertices with identifier id and state $s \in S = \{ENABLED, BLOCKED\}$. Each vertex corresponds to exactly one component in a composite application.
- $E = \{e|e = (v_1, v_2, s, t)\}$ set of edges corresponding to possible communication paths between components $v_1, v_2 \in V$ with state $s \in S$ and with textual annotation t.

A data flow restricted by the model $G = (V, E)$ takes place as follows:

- A component corresponding to the vertex v is allowed to emit or receive messages only if $v.s = ENABLED$.
- A message m from a component corresponding to v_1 is allowed to be delivered to a component corresponding to v_2 only if $\exists e \in E : e = (v_1, v_2, ENABLED, t)$.
- The data flow takes place according to the original messaging pattern if none of the above restrictions apply.

The model is visualized and can be modified by dedicated UI extensions (cf. Figure 5.6).

- States $s \in S$ of vertices are visualized using borders of different color and type around the corresponding components.

- Potential communication paths $e = (v_1, v_2, s, t) \in E$ are visualized using arrows between components corresponding to v_1 and v_2. The arrow style indicates the state of the communication path $s \in S$. Annotation t is displayed above the corresponding arrow to provide additional information on the communication path. In case of topic-based publish-subscribe communication, name and description of the topic are displayed.

- For every vertex v, visualization of its state $s \in S$ and in-/outgoing edges $e = (v, *) \in E$ can be turned on or off to avoid cognitive overload in case of strong connectivity.

- For every vertex v, its state $s \in S$ can be toggled through the components configuration menu.

- For each edge $e = (v_1, v_2)$, its state $s \in S$ can be toggled by clicking on the corresponding arrow.

Two architectural extensions have been introduced to implement management of communication models. The Awareness and Control Module has been added to Apache Rave and is responsible for synchronization among the communication model and the WebComposition/EUD composition model. To derive a communication model out of a composition one the Algorithm 5.1 is applied.

The algorithm is executed on every change of the composition model. It starts with an empty graph and adds a vertex for each component of the application. States of vertexes are set to $BLOCKED$ or $ENABLED$ depending on if components are isolated in the composition model. The edges between the vertexes are derived out of publications and subscriptions common to pairs of components and eventual transformation functions. If communication between two components over some topic is restricted, the state of the corresponding edge is set to $BLOCKED$.

Algorithm 5.1: Creating a Communication Model out of a WebComposition/EUD Composition Model

Input : WebComposition/EUD Composition Model as defined in Section 4.3.2

Output : Communication Model $G = (V, E)$

1. Set $V = \emptyset$, $E = \emptyset$
2. For each component $c_i \in C$, create a new vertex $v_i = \langle i, s_i \rangle$ with $s_i = ENABLED$ if $v_i \notin I$ and $s_i = BLOCKED$ otherwise. Set $V = V \cup v_i$. In the following, component c_i will correspond to the vertex v_i and vice versa.
3. For each pair of components $\langle c_i, c_j \rangle$ and for each $p_{ik} \in c_i.PUB : p_{ik} \in c_j.SUB$, create a new edge $e_{ik} = (v_i, v_j, s_j, p_{ik})$ where $s_j = ENABLED$ if $c_i, c_j \notin I \wedge \langle c_i, c_j, p_{ik} \rangle \notin R$ and $s_j = BLOCKED$ otherwise. Set $E = E \cup e_{ik}$
4. For each pair of components $\langle c_i, c_j \rangle$ and for each $p_{ik} \in c_i.PUB : \exists f : t_g \longrightarrow t_h, t_g \in c_i.PUB, t_h \in c_j.SUB$, create a new edge $e_{ik} = (v_i, v_j, s_j, t_g)$ where $s_j = ENABLED$ if $c_i, c_j \notin I \wedge \langle c_i, c_j, t_g \rangle \notin R$ and $s_j = BLOCKED$ otherwise. Set $E = E \cup e_{ik}$
5. Return $G = (V, E)$

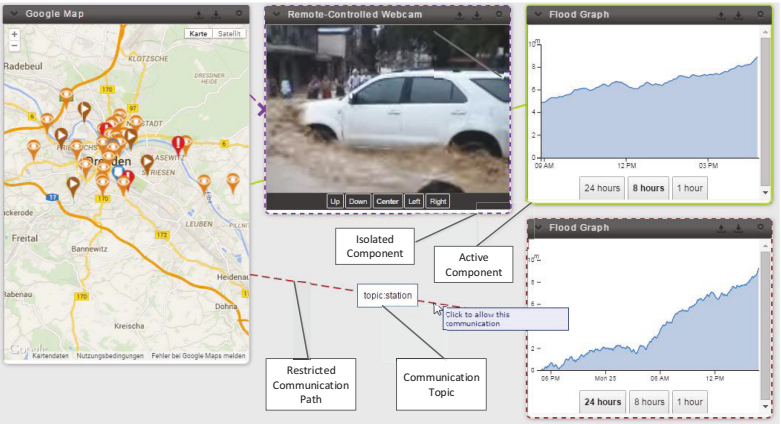


Figure 5.6.: Awareness and Control Facilities

The reverse mapping from the communication model onto the composition one is performed using the Algorithm 5.2.

The transformation is straightforward and creates isolations and restrictions for every vertex and edge with corresponding states.

To extend automatically derived communication paths, the transformation functions described in the Section 4.3.2 are used. The latter enable message exchange between components that do not share common publications and subscriptions. The definition of these channels takes place in a dedicated module called *Transformations Editor* (cf. Figure 5.7) (Schmiedel, 2013). The target group of Transformations Editor are Composition Developers with basic understanding of information representation or basic programming skills.

To define a transformation function Composition Developers first select a publication and a subscription of two different components. The function editor imports then corresponding topic definitions and visualizes their schemes. Users can then “map” data elements from one scheme

Algorithm 5.2: Deriving ICC Configuration out of a Communication Model

Input : Communication Model $G = (V, E)$
Output : Partial ICC Configuration $\langle R, I \rangle$ as defined in Section 4.3.2

1. Set $R = \emptyset$, $I = \emptyset$
2. For each vertex $v_i \in V : v_i.s = ISOLATED$, set $I = I \cup c_i$, where c_i is a component that corresponds to the vertex v_i.
3. For each edge $e_j = \langle v_{j,sender}, v_{j,receiver}, s_j, t_j \rangle \in E : s_j = BLOCKED$, set $R = R \cup \langle c_{j,sender}, c_{j,receiver}, t \rangle$
4. Return $\langle R, I \rangle$

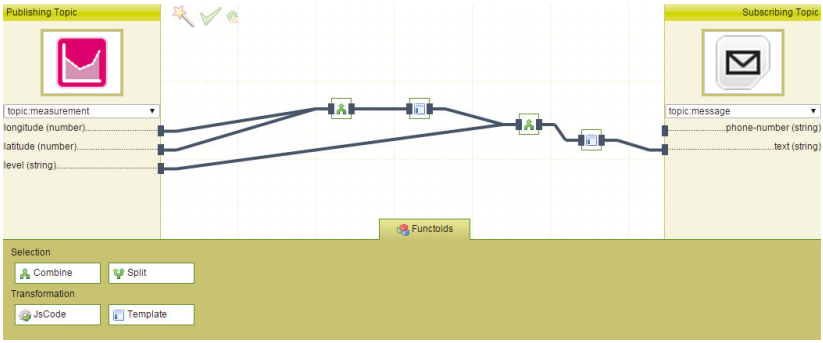


Figure 5.7.: Visual Definition of Transformation Functions

onto elements of the other and define optional transformations using so-called *Functoids*. The latter represent parameterizable string functions that accept a list of data elements as input and produce another list as output. The editor offers four types of functoids: *Concatenation*, *Splitting*, *Template Instantiation* and *Javascript Transformation*. The Concatenation functoid joins two inputs using a given separator string. The Splitting functoid performs the reverse operation and splits one data element into two using a given separator. Template Instantiation functoid replaces a placeholder within given template with the value of input element. Finally, Javascript Transformation enables arbitrary mapping from one input to one output element. Functoids can be combined into a pipeline to realize more complex transformations. The mapping defined in the editor is then transformed into an executable Javascript code acting as the final topic transformation function.

5.5 Evaluation

This section analyses to which extent the requirements from Section 5.2 are fulfilled by the implemented prototype.

Customization Support is a functional requirement that can be evaluated within acceptance tests with the platform. The composition platform has been installed and made available for public[5]. The corresponding extensions have been submitted as patches to the communities of respective open-source projects[6]. The requirement has been fulfilled completely: the Live Composition Editor enables full control of the composition model including metadata, components, component placement and communication configuration. All elements beside of viewport definitions can be visually customized by Composition Developers. Viewports and their attributes are dynamically managed by Apache Rave in the context of so-called templates. The latter can be assigned to the composition by Composition Developers.

[5] `http://vsr-demo.informatik.tu-chemnitz.de:8080/portal`, Retrieved: 22.05.2015

[6] `https://issues.apache.org/jira/browse/RAVE-831`, Retrieved: 22.05.2015

The *Ubiquity* requirement is implicitly fulfilled by choice of Web technologies for implementation and by responsive design techniques applied in visual editor of Apache Rave. To access the platform Composition Developers only need an Internet-capable device and a standard Web browser. No additional software installation or operating system is required.

The *Usability* requirement has been tested in the context of two user studies. The goal of the first study was to check if the proposed awareness and control facilities make Composition Developers more efficient in simple data collection and comparison tasks. Furthermore, usability of the applied concepts has been assessed based on user feedback. The second study tested suitability of the Transformation Editor for skills of Composition Developers. Its usability has been also evaluated based on user feedback.

5.5.1 Awareness and Control Facilities

To answer the question if the awareness and control facilities make users more efficient in simple data collection and comparison tasks, a laboratory experiment with 27 participants has been conducted. It results have been analyzed using statistical hypothesis testing (two-tailed t-test with significance level 95%) with null-hypothesis: *The average time for task completion using the awareness and control facilities is lower than the average task completion time without them*. Usability properties of the mechanisms have been assessed with average user ratings based on a 5-point Likert scale.

The user study has been conducted in cooperation with several research partners mainly T-Systems MMS[7] and TIE Kinetix[8] during the European FP7 Open Mashup Enterprise service platform for LinkEd data in The TElco domain (OMELETTE) project[9] (OMELETTE Consortium, 2013c). Complete evaluation material can be found in Appendix B.1).

[7] `http://www.t-systems-mms.com`, Retrieved: 1.11.2014
[8] `http://tiekinetix.com`, Retrieved: 1.11.2014
[9] `http://cordis.europa.eu/project/rcn/95584_en.html`, Retrieved: 29.5.2015

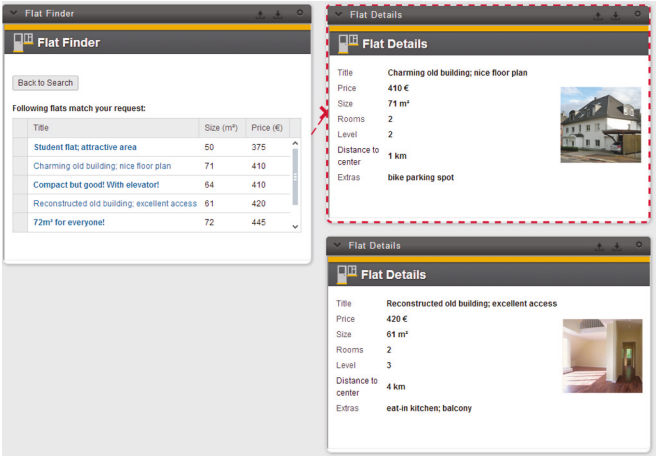


Figure 5.8.: Evaluation Application for Testing Usability of Awareness and Control Facilities

Setup 27 users participated in the study, 90% of which had no programming skills but were domain experts in marketing and telecommunication. The majority of users (74%) had an understanding of the term "widget", which was mostly related to mobile devices and the "Windows Vista Sidebar". Only 4 out of 27 users had ever configured a composite application (mostly intranet portals) on their own e.g., by repositioning of widgets and changing the color scheme.

Procedure The study applied laboratory experiment evaluation method. For each of the 27 participants, the evaluation procedure involved the following steps. Before the task execution, participants filled in a pre-evaluation questionnaire to judge their skill levels. Based on the results, they were evenly distributed over test and control groups. After that, users were given an introduction on the composition platform, its purpose, concepts and core functionalities. Following the introduction, users had the chance to explore and try out different aspects of the platform as they liked.

In the experiment, participants had to solve a comparison task using two components (cf. Figure 5.8). The context was given by the motivation scenario introduced in Section 2.1.3. Two component types were used: the one provided an overview of flat offers in the selected city and the other showed details of the offer. The task was to find the cheapest flat in one city and then a comparable one in another city. While the control group used the default setting with one "overview" and one "detail" component, the test group was provided with two "detail" components. The test group could use facilities for visualization of possible communication paths and isolate one component to simplify the comparison. Once isolation was enabled for a detail component, it would "freeze", so that new details could be loaded in the second component and easily be compared to the first one. The task completion time was measured for each user group. After the experiment users were asked to fill in a questionnaire and to assess the perceived ease of use and usefulness of the tools (cf. Appendix B.1).

Results The average time required to solve comparison tasks using the proposed measurements was slightly below the corresponding value of the control group (cf. Figure 5.9). The advantage is not statistically significant, so that the null hypothesis is rejected. Thus, there is not enough evidence to claim that the facilities make Composition Developers more efficient. However, the measurement is an indication for possible positive influence of the mechanisms on the task completion time.

The analysis of the post-questionnaire led to the following results. 64% of users disagreed or strongly disagreed that the proposed awareness and control facilities were cumbersome (cf. Figure 5.10, left). Some users recommended introducing more systematics in color assignments during visualization of communication channels. Others suggested making isolation controls accessible from the component header bar and to use alternative terms to "isolation" such as "fixing".

The vast majority of users (93%) found the awareness and control mechanisms useful (cf. Figure 5.10, right). One reason might be that users could accomplish the task in the workspace, without additional

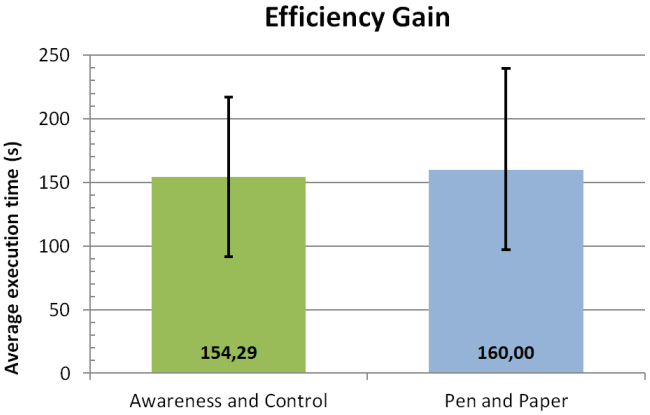


Figure 5.9.: Impact of Awareness and Control Facilities on Efficiency of Composition Developers

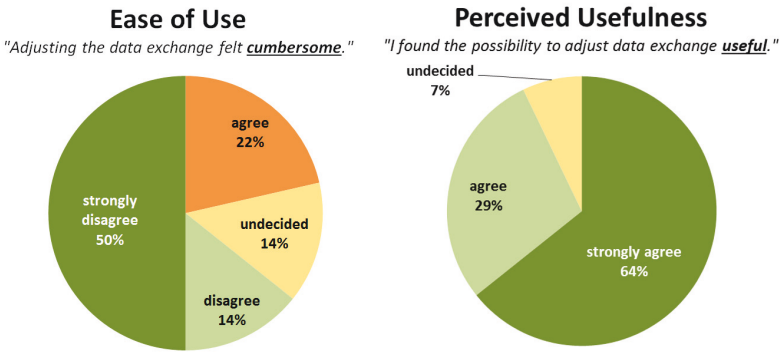


Figure 5.10.: Usability Evaluation of Awareness and Control Facilities

helpers, like pen and paper. Also, the cognitive load of comparing numbers in widgets directly on the screen is lower than switching back and forth between different media.

5.5.2 Transformation Editor

The goal of the second user study was to test suitability of the Transformation Editor for skills of Composition Developers and to evaluate its usability. The suitability has been assessed based on user-expressed confidence regarding application of the tool. Usability properties of the tool have been assessed with average user ratings based on a 4-point Likert scale. The complete evaluation material can be found in Appendix B.2.

Setup The study applied laboratory experiment evaluation method. 7 users with different programming skills participated in the experiment. 5 of them were non-programmers. However, 3 of them assessed their PC skills as advanced. 2 had experience in programming. The extended versions of Apache Rave and Wookie have been installed on a local machine.

Procedure Each participant received a short explanation of the Web-Composition/EUD concepts and of ICCI incompatibility problem. The functionality of the tool was not explained and had to be learned by participants themselves. Afterwards, users had to solve two tasks regarding data mapping between two components and fill in a questionnaire on different aspects of the system.

In the first task users had to create a new communication channel between a map and an address book component (cf. Figure 5.11, top). The first component could locate a given address on the map, while the other one displayed a list with addresses and published entry data upon mouse click. The goal was to display location of a selected address book entry on the map. The two components used different topics with incompatible schemes, so that a mapping between them was required.



Figure 5.11.: Tasks for Evaluation of Transformations Editor

The first task didn't require any Functoids to be used, but rather to provide a simple mapping between scheme elements.

In the second task users were asked to establish compatibility between the above address book component and a component that enabled creation of simple post cards (cf. Figure 5.11, bottom). The goal was to set recipient address of the post card using selected entry from the address book. Again, the interfaces were originally incompatible and had to be made so by users. The second task, however, required usage of several Concatenation and Template Instantiation Functoids.

Results The two tasks have been solved successfully by all study participants. In the post-questionnaire 86% of users confirmed or partially confirmed that they learned the Transformations Editor quickly (cf. Figure 5.12, left). The high rating indicates, that the UI of the tool was clear and self-explaining. 57% confirmed or partially confirmed that it was clear how to solve the task using the tool (cf. Figure 5.12, right). 72% confirmed or partially confirmed that the definition of mappings could have been simpler. Some users expressed doubts that they would be able to solve similar problems outside the experiment. Many users asked for assistance mechanisms or introductory videos to get more confidence and to learn the tool faster. Also they suggested different UI improvements such as more prominent symbols for Functoids, for their inputs/outputs and for intermediate results. The responses and requests for better assistance show that the participants were unconfident while applying the tool. This suggests that the proposed UI and interaction techniques do not significantly lower the complexity of data transformation task. Thus, the Transformation Editor matches skills of non-programmers insufficiently.

5.6 Summary

This chapter presented the WebComposition/EUD composition platform, which enables development and execution of component-based DSSs. Its conceptual architecture and implementation using Web technologies

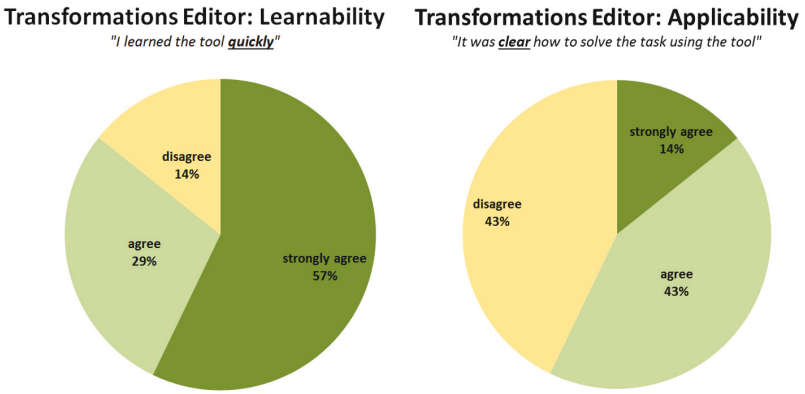


Figure 5.12.: Usability Evaluation of the Transformations Editor

have been described. Evaluation of the proposed DSS customization techniques during a user study showed feasibility and acceptance of the approach. It also provided valuable insights into possible improvements, especially related to visualization of component communication. The next two chapters present various extensions to the composition platform. Chapter 6 describes mechanisms to speed-up development of DSSs, while Chapter 7 focuses on evolution and management activities of software artifacts.

6 Development Assistance

The composition platform presented in Chapter 5 enables development and customization of component-based DSSs. Under time-pressure, however, the development process has to be accelerated. This chapter presents three mechanisms that address this goal. The first one, Automatic Discovery and Composition Engine, is intended to speed-up composition development by suggesting solutions that might fit user goals best (cf. Section 6.2). The two others improve reliability and efficiency of solutions during their usage: Loop Detection Facilities detect potentially faulty configurations that might harm the run-time environment (cf. Section 6.3), and Double Input Detector automatically detects implicit relationships between components and automates user input within (cf. Section 6.4).

6.1 Research Questions

Despite of the simplicity of the WebComposition/EUD models, their discovery and composition still require manual effort. Mistakes in the configuration due to time-pressure can further delay the development process. The main challenge addressed by this chapter is how to speed up development of solutions and make it less error-prone. The research questions addressed are:

Research Question 1 Is it possible to speed-up development of WebComposition/EUD compositions by automating discovery and composition of components? This question analyzes if Composition Developers become more efficient with the proposed assistance mechanism.

Research Question 2 Under what conditions WebComposition/EUD compositions cause a looping message behavior and is it possible to detect the loops automatically? This question studies the causes and effects of looping messages and assesses the applicability of the proposed algorithms for early loop detection.

Research Question 3 Is it possible to automatically complete user inputs across WebComposition/EUD components based on history of repeated data entries? This question analyzes the suitability of the proposed algorithms and architectural extensions for automation of user inputs.

6.2 Automatic Discovery and Composition Engine

Development of component-based Web applications from scratch is a tedious and time-consuming process. Composition Developers have to find, understand and appropriately configure components to achieve their business goal. The WebComposition/EUD development process avoids building solutions from scratch and systematically supports reuse of existing artifacts. One of the activities of the Solution Planning step is to obtain a first DSS prototype that would partially fulfill requirements of Composition Developers, and then enable its iterative refinement through customization (cf. Section 4.4). The ADCE creates such prototypes on-the-fly based on its knowledge base and a dialog with the user.

6.2.1 Motivation Scenario

To illustrate the advantage of discovery and composition mechanism, consider a situation, where Peter, a coordinator of an emergency response team, wants to inform himself about rising water levels in his city and to decide on eventual rescue activities. He wants to create an application that would act as a DSS and would provide different kind of information for decision making. In particular, he needs an overview of water level measurement stations and their locations, measurements themselves and live views on different city areas. Additionally, he wants to contact other people in case further actions are required.

To create such a DSS using traditional composition environments Peter has to find and assemble appropriate components. Peter would use some sort of Component Browser and would search for components using keywords. Inspection of each result would be time-consuming - in many cases, textual descriptions are scarce or ambiguous. Nevertheless, Peter would go through the result sets and select components, he thinks would fit best. Having added them to the empty application Peter would find out, that some components do not have any inputs, but rather expect them to come from others, not yet present in the composition. After playing a bit with the composition, it would turn out that some components have incompatible ICCIs and do not exchange any information among each other. Peter would have to search for other combinations of components that would both provide the envisioned functionality and also be compatible in the composition.

The scenario shows, that finding and configuring a composite application can be a time-consuming and error-prone process. It would be desirable to support Composition Developers in this task and to let them start with existing implementations or patterns for common problems instead of building ones from scratch.

6.2.2 Requirements

The following specific requirements have been derived based on the goals of the Solution Discovery step and the described motivation scenario.

Automation Composition Developers should be able to obtain a first prototype of DSS without manual specification of its composition model.

Efficiency Development of WebComposition/EUD applications should be accelerated compared to the manual construction process.

Usability The assistance mechanism should be accepted by users and perceived as a useful tool that they also prefer to use.

The following section presents the idea and architecture of a mechanism that addresses the above requirements.

6.2.3 Automatic Discovery and Composition

The goal of the ADCE is to efficiently discover or produce a prototype of a DSS that would meet user goals as close as possible. To achieve this ADCE provides a dialog-based UI that enables interactive specification of business goals in form of a question-answer game. The produced goal model is then used to query Compositions Repository and internal knowledge base in order to obtain the best matching solution. Details of the discovery process are described in the next sections in more details.

The approach employs the definition of goals as targets for achievements that provide a framework for the desired system (Anton, 1997). Goals represent "high level objectives of the business, organization, or system" and "guide decisions at various levels within the enterprise". Examples of high-level business goals could be "to collect information

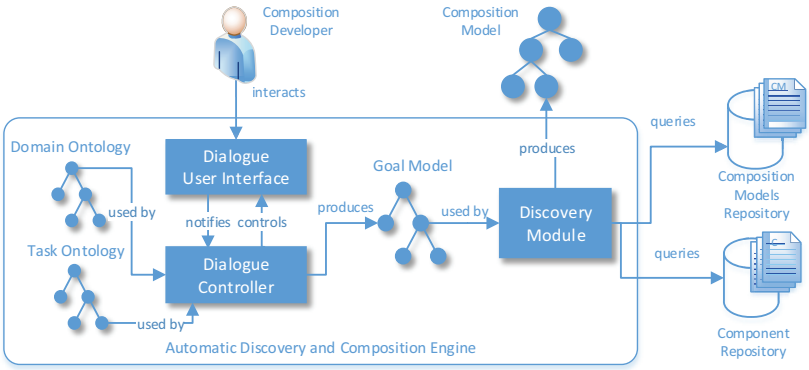


Figure 6.1.: Discovery and Composition Process

on emergency incidents", "to plan a trip" or "to organize a meeting". Due to the plenty of possible goals and ways to achieve them, ADCE refines initially specified business goals and proposes possible solutions based on internal knowledge base (cf. Figure 6.1).

Composition Developers interact with *Dialog Controller* through its UI. The latter uses *Domain* and *Goal Ontologies* to drive the conversation. The Domain Ontology describes concepts, their structure and relationships for a particular domain. It distinguishes between two top-level classes – Actions and Objects – and describes their subclasses. Figure 6.2 illustrates an excerpt of the Domain Ontology related to the emergency response scenario. The ontology defines a set of action ("Find", "Create", "Organize" etc.) and object classes ("Storm", "Flood", "Social Media" etc.). Each class is labeled with keywords that are used in natural language while referring to corresponding action or object. For example, the "Find" action class is labeled with keywords "show", "display" and "find". They keywords build the basis for dialog-based interaction with users.

The Goal Ontology contains knowledge about common business goals and their hierarchy. The ontology makes use of a single top-level class *Goal*, defines its instances and mutual relationships. Each instance is

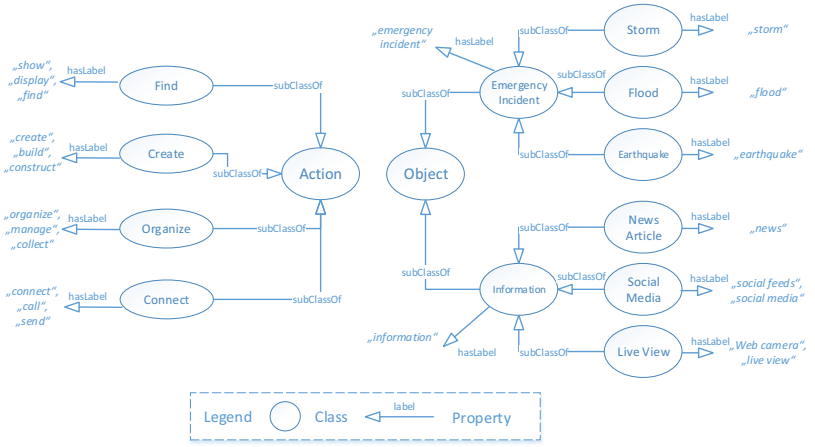


Figure 6.2.: Excerpt from the ADCE Domain Ontology

associated with one Action and one Object concepts from the Domain Ontology. Figure 6.3 illustrates an excerpt from the Goal Ontology related to the emergency response scenario. It defines among others the goal "Collect information on emergency incidents", which has corresponding relationships to the "Find" and "Emergency Incident" concepts from the domain ontology. Both ontologies are maintained by Artifact Library Manager and can be created either manually or semi-automatically by harvesting RDF data sets published on the Web.

The Dialog Controller lets user specify a goal and refines his selection in a number of question-answering steps. First, the user is asked to confirm the proposed goal decomposition. The decomposition can be adjusted or further subgoals can be added. Afterwards, the controller suggests refining Objects used in the subgoals. The result of the conversation is a model of user's business goal. It consists of one high-level goal coming from the Goal Ontology and its eventually customized decomposition. Each goal is represented through an Action-Object pair.

The model is used by *Solution Discovery Module* to find or produce solutions that would fit the stated goal best. The module makes

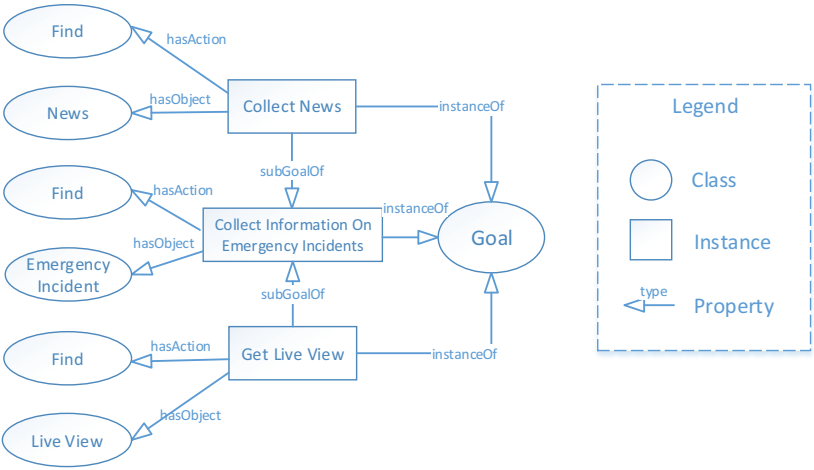


Figure 6.3.: Excerpt from the ADCE Goal Ontology

use of dedicated annotations of WebComposition/EUD Composition Models and Components. The annotations symbolized with $goals = \{\langle action, object \rangle\}$ are pairs that can be added by Artifact Library Managers to user solutions and available components. Query Builder applies Algorithm 6.1 to find an existing solution that would match the model of user's business goal best.

The algorithm searches for existing composition models that were annotated with the goal selected by user. In case several models are found, they are ranked based on included components – the more subgoals are fulfilled, the higher the rank is. The best ranked solution is then selected. The discovered composition model (also if empty) is then completed with components that address the yet unfulfilled subgoals. Algorithm 6.2 presents the applied completion algorithm.

First, subgoals are detected, for that no matching components are present in the discovered solution. Then the set of all available components is analyzed and the ones are selected that match the "unfulfilled" subgoals. In case one subgoal is matched by multiple components, those

Algorithm 6.1: Discovery of Solutions based on a Model of Business Goals

Input : Set of Annotated Composition Models $CC = \{cc_i\}$, Model of User's Business Goal $g = \langle g_{top}, \{g_{sub}\}\rangle$, Domain Ontology DO

Output : Composition Model CC_{New}

1. Find all $cc \in CC : g_{top} \in cc.goals$.
2. If no cc is found, return an empty model $CC_{New} = \langle g_{top}, g_{top}, \emptyset, \emptyset, \langle \emptyset, \emptyset, \emptyset \rangle \rangle$.
3. For each found cc_i calculate its rank being the number of subgoals g_{sub} matched by any component $c_j \in cc_i.C$. A component c_j matches a subgoal g_{sub} if $\exists \langle action^*, object^* \rangle \in c_j.goals : (g_{sub}.action = action^* \vee g_{sub}.action\ sameAs\ action^*) \wedge (g_{sub}.object = object^* \vee g_{sub}.object\ subclassOf\ object^*$. The relationships $sameAs$ and $subclassOf$ are discovered from DO.
4. Return cc_i with the highest rank.

Algorithm 6.2: Completion of Discovered Solutions

Input : Discovered Composition Model CC_{New}, Set of Available Annotated Components $C = \{c_i\}$, Model of User's Business Goal $g = \langle g_{top}, \{g_{sub}\}\rangle$

Output : Components to be Added C_{New}

1. Find all subgoals g_{sub} that are not matched by any component of CC_{New}. Use matching condition from Algorithm 6.1.
2. For each found subgoal $g_{sub,k}$ find all components $c_i \in C$ that match $g_{sub,k}$. Use matching condition from Algorithm 6.1. Let $C_k \subset C$ be a set of components matching $g_{sub,k}$.
3. $\forall k$ rank each component $c_{k,j} \in C_k$ with a number that corresponds to the number of $c_t \in CC_{New}.C : c_{k,j}.T \cap c_t.T \neq \emptyset$
4. Set $C_{New} = \emptyset$. $\forall k$ add one component from C_k with the highest rank to C_{New}. Return C_{New}.

components are selected that have the most "compatible" ICCIs with the ones from discovered solution. The latter is completed with new components and then loaded into Composition Platform for further customization by Composition Developers.

The implementation of ADCE is split into client and server parts. The client-side UI of the ADCE is implemented as a separate W3C widget, which communicates with server-side Dialog Controller using a RESTful API (cf. Figure 6.4).

Domain and Goal Ontologies are described in RDF and are parts of the Dialog Controller. The latter uses Apache Jena[1] framework for RDF processing. Annotations required for discovery and composition algorithms are stored in OMDL and W3C configuration documents in serialized form. OMDL places them into the *goal* XML-element, which already exists in the specification. Configuration documents of

[1] `https://jena.apache.org`, Retrieved: 5.3.2015

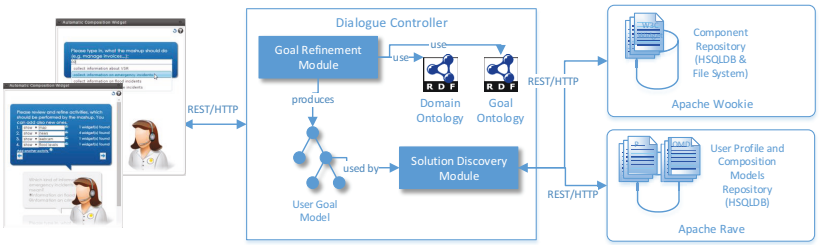


Figure 6.4.: Dialog-based Solution Discovery and Composition

WebComposition/EUD components are extended with dedicated *tag* XML-elements.

6.2.4 Related Work

Several approaches that simplify or automate composition of component-based applications has been proposed. The one group, which can be referred to as bottom-up composition engines, automates assembly of components without taking high-level user goals into account. In (Ngu et al., 2010) authors propose a framework enabling a progressive composition of portlets based on semantic annotations and dedicated matching algorithms. The system requires its users repeating search requests to find every next component and doesn't focus on high-level business goals. (Roy Chowdhury, Rodríguez, et al., 2012) describes a similar system, whose purpose is to analyze current constellation of a composite application and to suggest further possible components and configurations. Recommendations come from a repository with mined composition patterns. Again, high-level business goals are not taken into account.

Another group of approaches performs a top-down application composition, starting with high-level description of user goals. In (Pietschmann, Radeck, et al., 2011) the user goal is described by a static template containing abstract component descriptions. Based on the template, the composition engine performs context-aware discovery, selection and

integration of component implementations. Authors do not describe the template specification process, however. In (Tietz et al., 2011) authors propose a task-based component discovery approach. A lightweight task ontology is introduced and a matching algorithm based on semantically annotated components is proposed. The starting point of the discovery algorithm is an instance of the semantic task model. Authors do not describe how non-programmers can produce such an instance.

6.2.5 Evaluation

In the following the approach is evaluated against the requirements from section 6.2.2.

Automation The presented approach doesn't require manual specification of composition models. Instead, they are discovered or composed automatically based on internal knowledge base and available artifacts. In case goal is not captured in the Goal Ontology users can select a similar one and refine its subgoals manually.

Efficiency and Usability The two properties have been evaluated in the context of the user study presented in Section Section 5.5 (OMELETTE Consortium, 2013c). The goal of the evaluation was to answer the question, if the ADCE speed-ups development of WebComposition/EUD compositions, and to assess its usability based on user feedback.

For this purpose a user study (laboratory experiment) with 27 participants has been conducted and its results have been evaluated using statistical hypothesis testing (two-tailed t-test with significance level 95%). The null hypothesis was: *The average time required for developing described solutions using ADCE is smaller than the average time required for developing the same solutions without the help of the tool*. Usability properties of the tool have been assessed with average user ratings based on a 5-point Likert scale. The complete evaluation material related to this experiment can be found in Appendix B.3.

Setup The same 27 users as in the user study presented in Section 5.5 took part in the experiment. The experiment used a local installation of the composition platform with an activated ADCE module.

Procedure First, users filled in a questionnaire, where they expressed their skills and familiarity with mashup and widget technologies. After that, users were introduced into the concepts of composition platform and the ADCE and could try out the tools as they liked. Then a test and a control groups with even distribution of skills, age and gender were built. The participants were asked to complete a task that required them to build a mashup consisting of three components. While the test group used ADCE, the control group created a composite application manually using the platform built-in facilities. The time required to accomplish the task for each group was measured. Finally, users filled a post-questionnaire, where they could assess usefulness and usability of ADCE using a 5-point Likert scale.

The following task was given: "Imagine you live in the city of 'Dresden' and there is a flood warning. Build a workspace to gather more information about the situation: (a) list messages from your social networks, (b) check where emergency incidents were reported, and (c) compare the flood levels at these locations" (OMELETTE Consortium, 2013c).

Results Users that were using the ADCE to build a solution were slightly faster than the ones, who used the platform built-in facilities (cf. Figure 6.5). The time spent was 15 seconds shorter - the difference is not statistically significant though (the null hypothesis is rejected). There is not enough evidence to claim the speed-up. The measurements, however, indicate a positive influence of automatic discovery and composition facilities on the overall development speed.

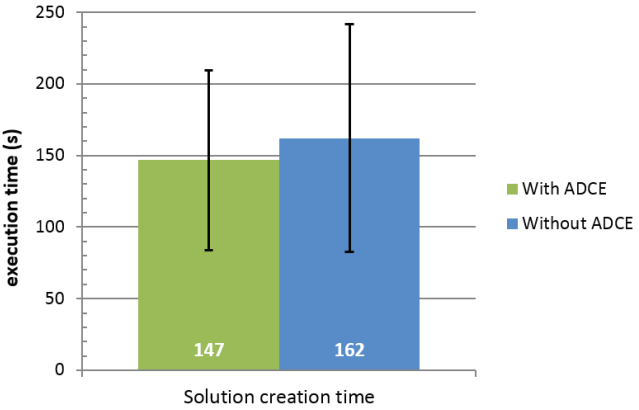


Figure 6.5.: Acceleration of Mashup Creation (Source: (OMELETTE Consortium, 2013c))

Most of the participants liked the idea of guidance during the composition process (79% agreed or strongly agreed, cf. Figure 6.6).

93% of participants agreed or strongly agreed that the ADCE is a useful mechanism (cf. Figure 6.7, left). Finally, 86% of users agreed or strongly agreed that they would use the ADCE again for building mashups; 14% were undecided (cf. Figure 6.7, right).

Regarding the users in the control group, which created solution manually, the majority of them (92%) found it easy to find appropriate components to fulfill their task. However, only 69% of users were sure that the widgets were the right ones. These finding underlines the necessity of user assistance during the composition process.

User study results indicate that non-programmers like interactive dialog-based interfaces. This finding should be considered while developing future end-user-oriented programming environments. The evaluation didn't show a significant time advantage while using ADCE. For the one it might be caused by several usability issues

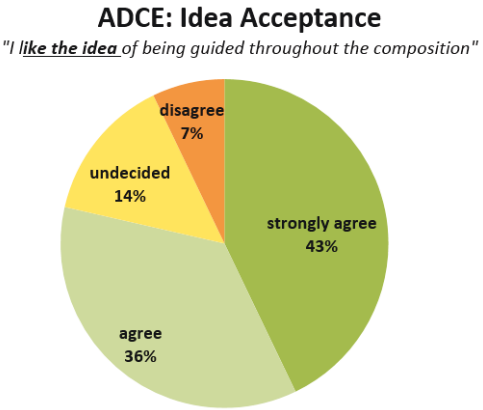


Figure 6.6.: Acceptance of the Guidance Idea

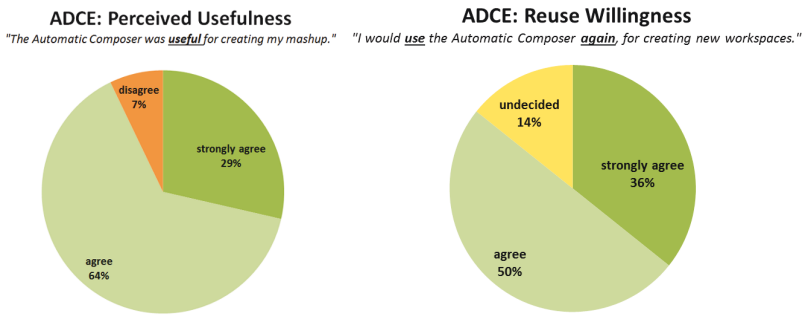


Figure 6.7.: Perceived Usefulness (Left) and Reuse Willingness (Right) of ADCE

and bugs discovered in the tool during experiments. Furthermore, the algorithm of discovery and composition wasn't explained clear enough, so that some users applied the tool inadequately. UI seems to play a very important role in performance-related tasks. Improper UI can negate time advantages gained by automation mechanisms.

6.3 Loop Detection Facilities

The WebComposition/EUD composition model foresees "self-organization" of components regarding their messaging behavior. While this behavior doesn't require any explicit configuration by users, an uncontrolled and, thus, unreliable communication can emerge. This can lead to time-consuming or cost-causing effects that are especially critical under time pressure. This section presents an assistance mechanism that analyzes structure and behavior of an WebComposition/EUD application and detects eventual problems in configuration.

6.3.1 Motivation Scenario

Consider the slightly modified example of application from Section 4.3.2 that produces an undesired loop behavior. The application aggregates information on emergency incidents caused by flood. Additionally to already included components it adds a new one, $c_{AllStations}$ that lists passed water measurement stations in a table and highlights the one that is closest to the user. The data to display should come as a notification message on topic $t_{StationList}$. Coordinates of the station closest to current user are automatically computed and published on topic $t_{Location}$. ICC capabilities of the map component are extended to work with the new component: once focus of c_{Map} is changed (either by user or by incoming messages with new focus coordinates), details to all water measurement stations around the centered area are published on $t_{StationsList}$. Although not explicitly configured, this application produces a self-reinforcing loop: once the center point of

c_{Map} is changed, a message to $c_{AllStations}$ is published, which, in its turn, produces a message on $t_{Location}$ with coordinates of the closest station. This message is received by c_{Map}, which moves its focus to the new location, and the communication starts for anew. Depending on the concrete run-time environment, the application can become slow, stop responding or even crash.

To avoid situations like the above, both the run-time environment and components should be protected from self-reinforcing loops. In the following, appropriate mechanisms are presented in details.

6.3.2 Requirements

Based on the above problem description and motivation scenario, we first elicit requirements on facilities that should help improve reliability of WebComposition/EUD applications.

Loop Discovery Messaging behavior that can cause uncontrolled and "looping" communication, should be automatically detected. The detection should take place as early as possible best before the application is instantiated and executed.

User Control Users should be notified and decide on the feasibility of potentially harmful WebComposition/EUD configurations. The goal is to enable communication behavior constructed by intent but to warn users about its possible consequences.

Efficiency Discovery of "looping" communication should be performed efficiently. In terms of algorithmic complexity a linear or at least quadratic complexity is desirable.

6.3.3 Loop Discovery

Before presenting the devised discovery algorithms and their implementation, several assisting definitions and formalization of the problem is introduced.

Terminology

Let $cc = \langle M, C, VP, CP, ICC \rangle$ be a WebComposition/EUD composition as defined in Section 4.3.2 and $a = \langle c_{sender}, t, data \rangle$ with $c_{sender} \in C, a.t \in c.PUB$ being some notification message.

Definition 6.3.1. A notification message a is called *user-triggered* if it was issued as a result of user interaction with c_{sender}.

Example of a user-triggered message is the one from c_{Map} to $c_{AllStations}$ containing details on water measurements stations around some area. The message is issued immediately after user moves the focus of the map.

Definition 6.3.2. A publication message a is called *subscription-triggered* with trigger a' (or triggered by a') if c_{sender} automatically issued a after receipt of the publication message a'.

In the above example messages issued by $c_{AllStations}$ are all subscription-triggered. They are issued as soon as messages with measurements stations from c_{Map} are delivered to the component.

Definition 6.3.3. A publication message a is called *internally-triggered* if it has been issued by c_{sender} independently of any prior interaction with user or receipt of any other message a'.

Internally-triggered messages are e.g., timer events or push messages from remote services. For instance, $c_{AllStations}$ could issue updated data

of closest water measurement station every 10 minutes without any explicit user request or incoming notification messages.

According to the above classification, one observes, that the unwanted behavior as described in Section 6.3.1 is caused by components that automatically issue responses to incoming messages i.e., produce subscription-triggered messages. On the other hand, the duration of such "looping" communication is unpredictable as any component can change its behavior at any point of time depending on the internal logic. These observations lead to the following two definitions of loops, which build a basis for subsequent discovery algorithms:

Definition 6.3.4. A *run-time* loop L_a in the composition cc is a message sequence $L = (a_0, a_1, ..., a_{n-1})$ so that $a_i, i = 1..n-1$ is subscription-triggered by a_{i-1} and $\forall i = 0..n-1 : (a_i.t \in a_{i+1\ mod\ n}.c.SUB \vee \exists t' \in a_{i+1\ mod\ n}.c.SUB : \exists f : a_i.t \longrightarrow t' \in F \wedge \langle a_i.c, a_{i+1\ mod\ n}.c, t' \rangle \notin R) \wedge \langle a_i.c, a_{i+1\ mod\ n}.c, a_i.t \rangle \notin R \wedge a_i.c \notin I$.

The definition focuses on message sequences that have been produced by "compatible" components and where the sender of the first message becomes also the receiver of the last one. Though the receipt of the last message doesn't necessarily mean, that the component will start a new iteration (it depends on its internal logic), the idea is to warn users of the potentially harmful situation and let them decide if the detected behavior is desirable or not. In the scenario from Section 6.3.1 a run-time loop could be a sequence from $a_0 = \langle c_{Map}, t_{StationsList}, data_{SurroundingStations} \rangle$ and $a_1 = \langle c_{AllStations}, t_{Location}, data_{ClosestStation} \rangle$. The loop emerges as soon as user moves the map focus to new location.

Definition 6.3.5. A *design-time* loop L_p in a composition cc is a sequence of components $L_p = (c_0, c_1, ..., c_{n-1})$ so that $\forall i = 0..n-1 : c_i \notin I \wedge (\exists t \in c_i.PUB : t \in c_{i+1\ mod\ n}.SUB \wedge \langle c_i, c_{i+1\ mod\ n}, t \rangle \notin R) \vee (\exists t_1 \in c_i.PUB, t_2 \in c_{i+1\ mod\ n}.SUB : \exists f \in F : t_1 \longrightarrow t_2 \wedge \langle c_i, c_{i+1\ mod\ n}, t_2 \rangle \notin R)$.

A design-time loop represents a sequence of senders that design-timely can produce a run-time loop at composition execution time. It con-

sists of non-isolated components that have non-blocked communication paths in the order defined by the sequence. Existence of a design-time loop in a mashup doesn't mean, that a run-time loop will inevitably occur. Depending on the internal logic and a concrete internal state components in the sequence might never publish any messages or publish only user-triggered ones. Existence of a design-time loop is a necessary condition for existence of a run-time loop, not the sufficient one.

In the above example $(c_{Map}, c_{AllStations})$ constitute a design-time loop. The two components are able to communicate over the topics $t_{AllStations}$ and $t_{Location}$ and both produce subscription-triggered messages.

In the following, we decompose the problem of loop discovery into two distinct ones – detection of loops before they actually occur (design-time loop discovery) and at run-time based on the message traffic (run-time loop discovery).

Design-Time Loop Discovery

Discovery of design-time loops is based on analysis of WebComposition/EUD composition model. In particular, it focuses on possible communication paths that can be described by WebComposition/EUD communication model (cf. Section 5.4.4). The model is a graph, with components as nodes and possible communication channels as edges. Both nodes and edges can be put into *blocked* state, meaning that corresponding restrictions or isolations exist in the composition model.

To find design-time loops in a composition, the Tarjan's algorithm (Tarjan, 1983) is applied on the communication model (cf. Algorithm 6.3). The algorithm identifies all strongly connected components. The latter correspond to design-time loops by definition – they contain at least one directed path that starts and ends in the same node. This means, that a WebComposition/EUD component can potentially produce a message that can trigger further ones that end in the component itself.

Algorithm 6.3: Discovery of Design-Time Loops

Input : WebComposition/EUD communication model $G = (V, E)$ as defined in Section 5.4.4

Output : Set of design-time loops $P = \{(c_{i1}, c_{i2}, ..., c_{in})\}$

1. Remove vertices $v_i \in V : v_i.s = ISOLATED$ from V and corresponding in-/outgoing edges from E.
2. Remove edges $e_j \in E : e_j.s = BLOCKED$ from E.
3. Remove all vertices $v_i \in V$ without any in-/outgoing edges.
4. Apply Tarjan's algorithm (Tarjan, 1983) to the graph G:
   a) Let $P = \emptyset$ be a set of all identified strongly connected components.
   b) Mark all vertices in the graph as not-visited. Let S be a stack that holds all visited vertices ($S = \emptyset$ at the beginning). Let $index_j \; \forall v_j \in V$ be the depth-search visit time of the vertex v_j and $lowlink_j \; \forall v_j \in V$ be the smallest index (depth-search visit time) of the vertex that is reachable from v_j.
   c) For each not-visited vertex $v_i \in V$ perform the steps 4d - 4f
   d) Push the vertex v_i under consideration to S. Update the visit time $index_i$ of the vertex and set $lowlink_i = index_i$.
   e) For each neighbor v_j of v_i do:
      i. If v_j is not-visited, then perform steps 4d - 4f for v_j (recursive call). Set $lowlink_i = MIN(lowlink_i, lowlink_j)$ afterwards.
      ii. If v_j is visited and $v_j \in S$, then set $lowlink_i = MIN(lowlink_i, index_j)$
   f) If $lowlink_i = index_i$, then set $P = P \cup (v_i, v_{i1}, v_{i2}, ..., v_{in})$, where $v_{i1}, v_{i2}, ..., v_{in}$ are the elements above v_i on the stack S. Remove $v_i, v_{i1}, v_{i2}, ..., v_{in}$ from S.
   g) Replace each vertex in P with the corresponding WebComposition/EUD component and return P.

The result of the algorithm is a set of discovered design-time loops.

Run-Time Loop Discovery

Discovery of run-time loops is based on "traffic" produced by communicating components. Components do not need to be aware of the discovery facilities - neither source code nor metadata have to be adjusted. Issued notification messages are stored in the order of appearance in a buffer and the buffer is analyzed using the Algorithm 6.4 after each new incoming message. The algorithm returns *True* if the newly added message yields a run-time loop and *False* otherwise.

Algorithm 6.4: Discovery of Run-Time Loops

Input : Composition model cc, notification message sequence $S = (a_1, a_2, ..., a_n)$ in order of message appearance

Output : $L_a = \{a_{i1}, a_{i2}, ..., a_{ik}, a_n\}$ being a run-time loop if the message a_n yields one, and $\emptyset$ otherwise

1. Construct a directed graph $G_c = (E_c, V_c)$ (so called *causality graph*) using the following steps:
   a) Let $V_c = \{a_n\}, E_c = \emptyset$.
   b) For each $a_i \in S \backslash \{a_n\} : (a_n.t \in a_i.c.SUB \wedge \langle a_i.c, a_{i+1\ mod\ n}.c, a_n.t \rangle \notin R \wedge a_i.c \notin I) \vee (\exists t' \in a_i.c.SUB : \exists f : a_n.t \longrightarrow t' \in F \wedge \langle a_i.c, a_{i+1\ mod\ n}.c, t' \rangle \notin R)$ add a_i to V and add the directed edge (a_n, a_i) to E_c.
   c) For each $a_i, a_j \in S : a_i\ subscription - triggered\ by\ a_j, i < j$:
      - Add a_i and a_j to V_c (if the vertices do not exist yet)
      - Add the directed edge (a_i, a_j) to E_c.
2. Using a depth-first search with a_n as a start node, check if a_n belongs to a circle in G_c. If a circle containing a_n exists (meaning the message a_n causes a run-time loop), return the circle nodes in the traversal order. Return $\emptyset$ otherwise.

The above algorithm spans a directed graph on the sequence of recorded messages. Graph edges show 1) casual relationships between messages and 2) messages, whose senders can potentially trigger further communication after receipt of a_n. A circle in graph G containing a_n corresponds to the definition of a run-time loop.

The module for design-time loop discovery has been implemented as an extension to the Live Composition Editor of Apache Rave. It observes changes in the current composition model and re-executes Algorithm 6.3 on every change. In case design-time loops are found, a corresponding notification is shown to the user. User can then display communication model and isolate part of components.

The module for run-time loop discovery has been integrated into the OpenAjax Hub and executes the algorithm 6.4 on every new message. Due to the missing information on the origin of notification messages (user-, subscription-, or internally-triggered one), the module applies a heuristic to distinguish between the different message types: a message a_j is assumed to be subscription-triggered by a_i, if $(a_i.t \in a_j.c.SUB \vee \exists t' \in a_j.c.SUB : \exists f : a_i.t \longrightarrow t' \in F) \wedge \langle a_i.c, a_j.c, a_i.t\rangle \notin R \wedge a_i, a_j \notin I \wedge T_j - T_i < \varepsilon$ with T_i, T_j being occurrence times of a_i, a_j correspondingly and ε – a fixed time threshold. The prototype implementation uses 10ms as value for ε and has been selected empirically. All other messages are considered to be user-triggered. The time threshold should ensure, that the second message was not produced as the result of user-component interactions. The latter is unlikely to happen within the short period. However, faulty classification is still possible. If a run-time loop has been detected, the module issues a warning to user with details on the loop. Any communication between components is stopped for the time of decision making. Users can either ignore the warning (in this case components causing the loop are added to a white list and no further warnings are issued) or forbid the communication (in this case isolations for loop-causing components are automatically added to the composition model).

6.3.4 Related Work

Related work on the topic may be found in self-organizing systems, where communication is not controlled by any single entity and where undesired effects such as looping messages or self-reinforcing communication can emerge.

For example, loops in E-Mail communication can be discovered using statistical analysis of message chains. In (Solana et al., 1996) authors introduce a heuristic function that detects suspicious messages based on a historical information about their occurrence. The function takes number of occurrences, occurrence time and time lapses between occurrences into account. Authors claim a "remarkable accuracy" of the loop detection algorithm. While the approach is applicable to find runtime loops, it requires rich statistical information to perform well. For time-pressuring situations early detection is desirable to avoid possible side-effects or even costs caused by implicit ICC configuration.

In the field of parallel programming the so called Livelock situation is very close to the described problem. A livelock is a state of a system, in that it remains active but doesn't make any progress in executing its tasks (Tai, 1994). This might be caused due to communication overhead, where two systems exchanges too many coordination messages among each other and are too busy to continue with their respective tasks. The problem is usually tackled with explicit state model checking techniques for finite state systems(Dong et al., 2003). Appropriate methods have been introduced for systems utilizing asynchronous message-passing paradigm for communication (Leue et al., 2006). In both cases, the livelock prevention is based on analysis of system's internal logic specification, i.e., source code, which is inaccessible for utilized application and component models. The proposed detection facilities are "non-invasive", i.e., the analysis takes place using composition model and issued messages only.

6.3.5 Evaluation

In the following, the approach is evaluated based on the requirements stated above.

Loop Discovery The proposed mechanisms are able to automatically detect so-called design-time and run-time loops. The design-time ones are discovered by analyzing the communication model, which describes potential message flows. Though design-time loops do not necessarily imply existence of run-time ones, a timely isolation of the affected components prevents their occurrence. The algorithm for run-time loop discovery is based on analysis of history of issued messages. It is executed after each new notification message and issues warnings if sequences satisfying definition of run-time loops are discovered.

Current prototype has several limitations. Under certain circumstances it produces both false negative and false positive errors. First, it doesn't discover "long-lasting" run-time loops, i.e., the ones, where time difference between a subscription-triggered message and its trigger is larger than the chosen threshold ε. Second, it discovers message sequences that might contain internally-triggered messages or even user-triggered ones, if they occur within ε from the last message publication. The reason for the errors is the absence of information on the cause of messages – the utilized component, composition and message models do not provide such data explicitly.

User Control LDF provide notifications and means for control of how the system should proceed in case run-time or design-time loops are detected. Until user meets a decision, the modules follow a defensive strategy – the communication gets blocked until it is enabled again by the user. The blockade of the communication is done by changing states of components in the WebComposition/EUD communication model.

Efficiency The efficiency of the both algorithms is evaluated in terms of algorithmic complexity.

The run-time loop discovery algorithm takes n messages as input. The construction of the causality graph takes place in $O(n^2)$ time as maximal all pairs of messages are considered. The subsequent depth-first traversal starting with the last message is performed in maximal $O(|E|)$ steps, where E the set of edges in the graph ($|E| < n^2$). The summarized complexity doesn't exceed $O(n^2)$ and, thus, satisfies the original requirement.

The design-time loop discovery algorithm takes a communication model consisting of $|V|$ vertices and $|E|$ edges as input. The clean up of the model, i.e., removal of isolated components and blocked communication paths is performed in $|V| + |E|$ steps. The complexity of Tarjan's algorithm is $|V^*| + |E^*|$ with V^*, E^* being the sets of vertices and edges after the first clean up steps. In summary the complexity of the overall algorithm is $O(|V| + |E|)$ and, thus, also satisfies the efficiency requirement.

6.4 Double Input Detector

In time-pressuring situations manual interactions with component-based DSSs can become error-prone and time-consuming. If aggregated components have incompatible ICCIs and do not exchange data automatically, users have to either specify dedicated transformation functions or manually synchronize component views (e.g., by copying form inputs). The both are time-consuming operations and are not feasible to be performed under time-pressure. It is required to support users in usage of component-based DSSs and to make them efficient while interacting with aggregated components.

6.4.1 Motivation Scenario

To illustrate the problem consider the example application from Section 4.3.2. Additionally to the map with water measurement stations and visualization of their data, the application should show weather forecast for location selected on the map. A user adds a new component that queries temperature and rainfall data for given location. However, the component doesn't react to changes in the map component – the map doesn't publish search queries entered into its search field. Also the map doesn't react to selections made in the forecast component – the topics used to identify display locations are different. If user doesn't have time or skills to resolve incompatibility issues, he has to retype or copy-paste the input in both components. The latter is at least time-consuming and can lead to typos under time-pressure.

It is desirable that the system unburdens users from repeated input and provides automation facilities for this purpose. For the above example, it would be desirable to automatically pass the entered location between the two components.

6.4.2 Requirements

In the following, requirements on the solution for the above problem are derived.

Input automation The system should enable automation of repeated input for two or more components. Data entered into one component should be immediately passed to others.

Personalization Each user should be able to decide independently if and how the input automation should take place in the current context. In the example above, users might have different expectations on the synchronization behavior. While the ones would like the location to be synchronized between the two components, the others can prefer only one direction.

Simplicity No programming skills should be required to apply the solution. Well-known UI concepts and control elements should be used to define, modify and remove the automation rules. The system should support end-user awareness and continuously provide feedback on its current state.

Efficiency Definition and execution of automated behavior should take place efficiently. Composition Developers should be able to define the desired behavior fast, so that the advantage of investing attention into the proposed assistance mechanism exceeds the temptation to perform all actions manually.

6.4.3 Automation of User Input

The presented approach makes use of the PBD technique (Lieberman, 2001) to produce behavior specifications by providing examples of the desired actions. The system generalizes the provided samples and applies them automatically to other similar contexts. The changes/inputs should be demonstrated using GUI of components only, so that users do not need to learn any modeling constructs but rather can reuse well-known and already understood interaction methods.

Before presenting the facilities to automate user input, we first introduce several assisting definitions.

Terminology

The following definitions describe user actions that can be automated by the proposed mechanism.

Definition 6.4.1. A *user input action* a in WebComposition/EUD component c is a tuple $\langle e, t_a, v \rangle$ with

- $e = \langle id, t_e \rangle, e \in c.GUI$ being an input element with unique name id and type $t_e \in T_e = \{TEXT - INPUT, DROPDOWN, BUTTON\}$

- $t_a \in T_a = \{click, type, change, startdragging, enddragging\}$ being the type of user input action

- v being a new string value of the affected input element or $NULL$ if $e.t_e = BUTTON$

An example of the user input action is $a_{example} = \langle \langle$"$city - name$", $TEXT - INPUT \rangle, type,$ "$Chemnitz$"$\rangle$. Though current DOM specifications (Le Hors et al., 2004) foresee many other element and action types supported by Web browsers, the definition focuses on the most crucial input elements and user actions that are especially relevant for the scenarios of this thesis.

Definition 6.4.2. A *pattern for user input action* p in component c is a tuple $\langle e, t_a \rangle$ with $e \in c.GUI$ and t_a being input element and action type as defined in 6.4.1. We say, a pattern p *matches* the user action $a = \langle e, t_a, v \rangle$ if $p.e = a.e \wedge p.t_a = a.t_a$. A pattern sequence $(p_i), i = 1..n$ *matches* a sequence of user input actions $(a_j), j = 1..k$ if $n = k \wedge p_i$ matches $a_i \; \forall i = 1..n$.

An example of a pattern for the user input action $a_{example}$ is $p_{example} = \langle \langle$"$city - name$", $TEXT - INPUT \rangle, type \rangle$. The term pattern is used to classify similar user input actions that differ only in the value entered into an input element of a component. For example, the pattern $p_{example}$ would also match the user input action $a_{example2} = \langle \langle$"$city - name$", $TEXT - INPUT \rangle, type,$ "$Dresden$"$\rangle$.

Definition 6.4.3. Let $A*$ be a set of all possible user input action sequences. An *automation rule* AR for two components c_1, c_2 is a function $r : A* \longrightarrow A*$ that maps a given user input action sequence performed in component c_1 (so called *trigger actions*) to a new input action sequence to be immediately performed in component c_2 (so called *reaction actions*).

An example of an automation rule is the function $r_{example}$ that for one $TEXT - INPUT$-action in component c_1 produces a sequence of $TEXT - INPUT$-actions for component c_2 that copies the value entered into component c_1 into all $TEXT-INPUT$ elements in component c_2. Obviously, $r_{example}$ is of limited practical value as it overwrites all fields in c_2 with the value entered in c_1. Furthermore it considers only one user input in component c_1.

The challenge is, thus, to enable users to specify feasible automation rules in a convenient way and to develop an infrastructure for their management and execution.

Input Automation

The proposed mechanism observes user inputs in one component and automatically derives corresponding inputs for others. The demonstrated interactions can be describes using the following definition:

Definition 6.4.4. An *automation sample* S is a tuple $\langle T, R \rangle$ with $T = (a_{t,1}, a_{t,2}, ..., a_{t,n})$ being user input actions in component c_1 (called *demonstrated trigger actions*) and $R = (a_{r,1}, a_{r,2}, ..., a_{r,k})$ being user input actions in component c_2 (called *demonstrated reaction actions*).

An automation sample describes user input actions that user wants to be automated between two components c_1 and c_2. The demonstrated sample represents a result of execution of an automation rule that user intends to specify - the demonstrated pair is one known argument-value pair of automation rule function $r : A^* \longrightarrow A^*$. Given only one argument-value pair, one can define indefinitely many functions r that would map the given argument onto the given value but would differ in at least one other pair.

The intelligence to generalize the given automation sample and to define a feasible automation rule is put into the following guidelines:

1. Reaction actions should be performed only if *all* demonstrated trigger actions are repeated in component c_1, whereas the new values might differ from the originally demonstrated ones.

2. Relationships between input elements in two components are defined by equality of their values in the demonstrated trigger and reaction actions. The first element e_t affected in the demonstrated trigger actions, whose value is equal to the value of some element e_r affected in the demonstrated reaction actions, is considered to be the *value source* of e_r.

3. *All* demonstrated reaction actions are repeated in component c_2, whereas their values $a_{r,j}.v$ are set to the ones from value sources as identified in 2. If element has no source, it keeps its original value.

The Algorithm 6.5 takes the above guidelines into account and derives an automation rule based on a given automation sample:

The algorithm first computes a pattern sequence that matches given trigger actions T. The pattern is used to identify user inputs in component c_1 that are similar to the trigger actions in the sense of the guideline 1. Afterwards, it defines a value-source function g computed according to the guideline 2. Finally, a function (automation rule) is defined that maps given trigger actions onto reaction actions. All action sequences that are not matched by the derived pattern sequence, should be ignored, meaning they are not related to the demonstrated trigger actions. If a matching sequence is detected, it is mapped onto reaction actions in line with the guideline 3.

Implementation

The execution of algorithm 6.5 requires observation and manipulation of GUI controls inside of aggregated components. However, due to the black box nature of W3C packaged apps, it is impossible to directly

Algorithm 6.5: Deduction of an Automation Rule based on an Automation Sample

Input : Composite application cc, automation sample $S = \langle T, R \rangle, T = (a_{t,i}), R = (a_{r,j}), i = 1..n, j = 1..k$ for two components $c_1, c_2 \in cc.C$

Output : Automation rule AR

1. Let $P = (p_1, p_2, ..., p_n)$ be a sequence of patters with $p_i.e = a_{t,i}.e \wedge p_i.t_a = a_{t,i}.t_a, i = 1..n$. P matches T by the construction.

2. Let $g : \{a_{r,j}\} \longrightarrow \{p_i\} \cup \{NULL\} j \in 1..k, i \in 1..n$ be a so called value-source function that identifies sources of values for reaction actions in S. The function is defined as follows:

   a) $\forall j \in 1..k : \nexists i \in 1..n : a_{r,i}.v = a_{t,j}.v$ let $g(a_{r,j}) = NULL$.

   b) $\forall j \in 1..k : \exists i \in 1..n : a_{r,i}.v = a_{t,j}.v$ let $g(a_{r,i}) = p_{min(i)}$.

3. Let $A = (a_1, a_2, ..., a_m)$ be some arbitrary sequence of input actions in component c_1. The automation rule AR is defined as the following function:

   a) If P doesn't match A, return $\emptyset$. Otherwise $m = n$ by definition of a matching pattern sequence.

   b) Return $A' = (a'_1, a'_2, ..., a'_k)$ with a'_j being defined as follows:

      i. $\forall j \in 1..k \wedge g(a_{r,j}) = NULL : a'_j = a_{r,j}$

      ii. $\forall j \in 1..k \wedge g(a_{r,j}) = p_l : a'_j = \langle a_{r,j}.e, a_{r,j}.t_a, a_{min(i)}.v \rangle$ where $i \in 1..n \wedge p_l$ matches a_i

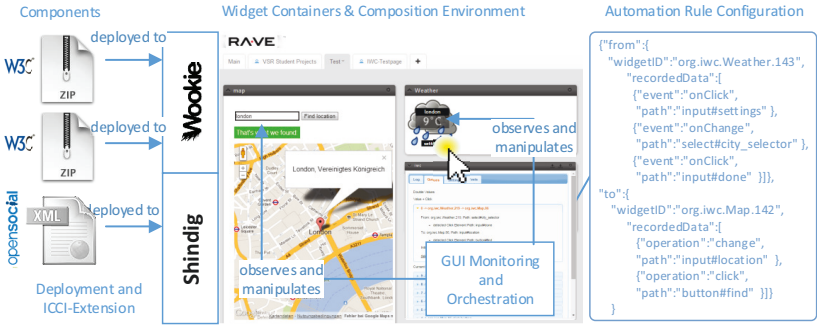


Figure 6.8.: Observation and Automation of Repeated Input

access their UI and logic at run-time. The only provided interface is the ICCI one. The idea is, thus, to automatically modify source code of components to enrich their ICCI interface with primitives for observation and manipulation of component GUI. Components should produce notification messages on GUI state changes and consume ones containing operations for modification of their GUI elements.

The modification of source code is performed automatically during deployment of a component to corresponding component container (cf. Figure 6.8). Apache Wookie has been extended to inject a piece of Javascript code into component packages that attaches DOM-event listeners to all detected input elements (cf. Definition 6.4.1). The injected code enables also manipulation of element values. State change notifications and manipulation requests are communicated using notification messages on dedicated "system" topics.

The Algorithm 6.5 is executed by a dedicated component being implemented as a standalone W3C packaged app (called *Controller Component*). The component should be part of the application to be able to observe and to replay user interactions. After an initial handshake phase, during which components from current composition model register themselves by Controller Component, the component starts observation of user interactions. Users do not need to trigger the learning process

explicitly - it starts automatically as soon as users start interacting with components. The Controller Component maintains a list of 10 last interactions and searches for automation samples. Current prototype supports only automation samples consisting of value changes with optional button clicks as trigger or reaction actions. If values of inputs in the trigger and reaction actions are equal, a new automation rule is created. The user is notified about the newly created rule with a pop-up window.

From then on, whenever a user starts interaction similar to the trigger actions with the source component, the system will automatically complete the corresponding interaction in the target component. The prototype stores automation rules in the Web storage of the browser (Hickson, 2013).

The Controller Component provides facilities for inspection and removal of the derived automation rule. The inspection enables highlighting of the affected input elements. Automation rules can be removed, if user is not interested in the behavior anymore.

The motivation scenario from Section 6.4.1 can be realized as follows. The two components are deployed into the system, where they are extended towards observation and modification of GUI. User creates a mashup out of the two components and adds the Controller Component. Afterwards, he demonstrates the desired behavior by selecting/typing and submitting some city name in both components. The controller component recognizes the given automation sample and derives an automation rule that maps selection and submission of any city name in the weather forecast component onto typing and submission of the same city name in the map component.

6.4.4 Related Work

Geppeto project introduced the idea of programming on the GUI level and applied it the context of widget-based dashboards (Skrobo, 2009). Using several special-purpose components and the PBD technique users

were able to define workflows consisting of multiple GUI actions across different widgets. However, the recorded workflows could only be triggered by user or by pre-defined system events and not by widgets themselves.

An approach for automatic synchronization of Web-based widgets is described in (Ghiani et al., 2011). Here, end users can interact with different Web applications aggregated on one canvas, while the data flow is recorded on a protocol level (e.g., parameters passed after a form submission). Afterwards, users are presented a dialog where they can establish connections between parameters of different requests, thereby denoting they share the same semantic concept. Every time an application then issues a server request, the system automatically submits connected applications, based on the "wired" parameters. As a result, reiterative manual data input is avoided and end users are presented a view of synchronized applications. The proposed solution works on the GUI level instead of the protocol one and derives relationships between "parameters" automatically.

Several research projects have focused on simplification of ICC configuration with the goal to automatically synchronize component states. For example, the drag&drop technique has been applied in the CRUISE project (Pietschmann, Voigt, et al., 2012) to establish ad-hoc connections between components. Interfaces have been extended with corresponding events that were triggered as soon as users dragged a piece of data outside of component boundaries. The platform offered compatible inputs of other components to receive the data and stored the assignment if desired. In the proposed solution a similar technique is applied with the difference that component extension is done automatically and synchronization of data takes place using GUIs of components.

6.4.5 Evaluation

In the following, the requirements from subsection 6.4.2 are reviewed and the approach is evaluated.

Input automation The approach enables automatic completion of user inputs across several components. Transitive relationships can be defined by demonstrating the automation behavior between pairs of components.

The proposed algorithm learns relationships between component input elements based on the history of entered data (equality of entered values). Automation rules are executed just after input sequences "similar" to the demonstrated ones are detected in source components. The algorithm has several limitations regarding recognizable relationships between input elements. It focuses on equality of values only and doesn't consider combination, splitting and transformation of input values. Specification of these more complex relationships is not feasible to under time pressure but is theoretically possible using transformation functions editor presented in Section 5.4.4. Execution of different reaction actions depending on the values of elements in the trigger actions is not supported yet and should be explored in future work. Another limitation of the implementation is the missing support for non-standard and dynamically added input elements such as ones implemented using various third-party Javascript libraries. The prototype relies on the HTML standard-based inputs available at the component instantiation time.

Personalization Personalization is achieved by enabling definition of automation rules for each user separately. To activate the automation functionality, users have to add the controller component to their applications. The configuration is stored within user browser and, thus, cannot interfere with configurations of other users. A drawback of this approach is that users are "bound" to one browser instance and cannot access the recorded configurations on other machines. An improvement of the current prototype would be integration of the automation rules into composition model and their persistence on the server side.

Simplicity Definition of automation rules doesn't require any programming skills and takes place automatically during users-application

interaction. Users are notified about discovered repeated inputs and can inspect involved elements in the controller component. Defined automation rules can be removed using dedicated control elements.

An open issue is, however, modification of automation rules. End-user-friendly editing of specifications produced by PBD method is in general complicated as it requires explanation of the derived generalizations (Lieberman, 2001). Currently, the editing is not supported – in case, an automation sample has been demonstrated wrongly, the user has to remove it and repeat the demonstration process from start.

Efficiency The definition and execution of automation algorithms takes place smoothly without any time delays for end users. The complexity of the automation rule learning algorithm is linear in the number of demonstrated actions. The construction of pattern sequence matching demonstrated trigger actions can be done in $|T|$ steps. The computation of value-source function pattern sequence can be performed in time $|T|+|R|$ if values of demonstrated trigger actions are pre-processed and information about their occurrence is memorized. Finally, the construction of an automation rule acting as a specification of the run-time behavior can be done in constant time.

Execution of automation rules is performed in linear time (considering sequence of input actions in some component as input). Input actions of length n are tested against the pattern sequence derived from demonstrated trigger actions in n steps. The application of value-source function to produce resulting reaction sequence requires maximal $|R|*n$ steps (depending on the number of non-empty input values in the original demonstrated reaction sequence).

6.5 Summary

This chapter presented three assistance mechanisms that aimed at assisting Composition Developers during the Component Discovery and Composition Development steps of the proposed DSS development process. The mechanisms have been integrated and evaluated within the WebComposition/EUD Composition Platform. The next chapter describes facilities that support evolution and maintenance of artifacts involved into the WebComposition/EUD development process.

7 Evolution Assistance

As presented in Section 4.4 the WebComposition/EUD development process starts with a Bootstrap phase, which concerns with analysis of the domain and initial provisioning of a set of reusable components for construction of DSSs. During evolution of resulting applications some of these steps are repeated – components have to be changed, their capabilities extended or new ones should be added. This chapter presents three assistance mechanisms that support Component Developers, Component Communication Experts and Artifact Library Managers in their respective tasks. The first tool, WebComposition/EUD-CC, enables efficient development of new functionalities by turning existing Web-based widgets into WebComposition/EUD components (cf. Section 7.2). The second tool, WebComposition/EUD-IE, supports modification and extension of existing components with new ICC capabilities (cf. Section 7.3). Finally, WebComposition/EUD Artifact Library (WebComposition/EUD-AL) acts a service that provides assistance in systematic management and efficient discovery of various WebComposition/EUD artifacts (cf. Section 7.4).

7.1 Research Questions

Capabilities of DSSs produced by end users are mainly defined by underlying WebComposition/EUD components. To enable development of DSSs for many use cases, it is important to provide a rich set of reusable components and enable their efficient discovery. This chapter addresses the challenge on how to efficiently create WebComposition/EUD components and how to enable their systematic management. The research questions addressed are:

Research Question 1 Is it possible to automatically transform Web-based widget packages of selected formats into WebComposition/EUD components? The question analyzes compatibility of WebComposition/EUD components with several legacy widget formats. Quality of the developed transformation algorithms is evaluated.

Research Question 2 Does the proposed visual environment speed-up the ICCI extension process? This question analyzes if the proposed tool makes Component Communication Experts more efficient in their tasks.

Research Question 3 Is it possible to perform centralized management of WebComposition/EUD artifacts using the Artifacts Library? The question analyzes if the proposed architecture and algorithms enable publishing, discovery and access control management of heterogeneous content.

7.2 WebComposition/EUD Component Converter

Effectiveness and acceptance of any composition platform significantly depends on the number and quality of components it provides. Without

a sufficient set of building blocks, Composition Developers are unable to construct useful and efficient support systems. Development of components, however, is a tedious and time-consuming task, especially if done from scratch. Although many repositories with reusable components exist on the Web, their reuse is hindered by incompatibilities in packaging formats, metadata descriptors and functional dependencies. It is required to support Component Developers and Artifact Library Managers in efficient provisioning and deployment of WebComposition/EUD components.

7.2.1 Motivation Scenario

Consider the example application from Section 4.3.2, which provides an aggregated view on emergency incidents around some location. Peter, a coordinator of a emergency response team, wants to add a component that would search for news from social media for a given keyword. However, he doesn't find any component with comparable functionality and performs a request to Artifact Library Managers to provide it. Although similar components can be found and downloaded from widget repositories on the Web, the components cannot be simply added to the WebComposition/EUD platform due to incompatible formats. Their manual transformation into WebComposition/EUD components or development of new ones from scratch are both costly and time-consuming.

In contrast, the development process would be much more efficient, if components of other types could be converted into the WebComposition/EUD format automatically. Artifacts Library can be then populated with popular and high-quality components from existing Web repositories, so that Composition Developers have richer choice of building blocks and fewer delays caused by development requests.

7.2.2 Requirements

Based on the above considerations the following requirements on automatic conversion mechanisms can be devised:

Compliance The assistance mechanism should produce WebComposition/EUD components out of existing Web-based components. Packaging format, dependencies and configuration documents should respect the WebComposition/EUD specification.

Efficiency The process of population should be efficient in terms of time and effort required. An optimal solution should be able to populate component library automatically, but at the same time provide sufficient control mechanisms of conversion process.

Extensibility The conversion mechanism should be applicable to more than one proprietary component format. The goal is to enable import of functionalities that users might have used before.

7.2.3 Conversion Process

The proposed tool enables transformation of proprietary Web widgets and gadgets into WebComposition/EUD components (Hertel, 2012). For this purpose, it defines a dedicated metamodel that acts as a common denominator between heterogeneous component formats. Adapters for different formats should produce package models that are then used to generate ready-to-use WebComposition/EUD components. Three adapters for different widget formats has been implemented: iGoogle gadgets[1], Opera Widgets[2] and Universal Web App (UWA) gadgets[3]. The conversion process foresees two steps: First, a widget model is derived out of a concrete packaging format and Second, a generic conversion

[1] `https://developers.google.com/igoogle/docs/igoogledevguide`, Retrieved 3.6.2015

[2] `http://web.archive.org/web/20130128182808/http://widgets.opera.com/de/`, Retrieved: 31.10.2014

[3] `http://uwa.netvibes.com/docs/Uwa/html/index.html`, Retrieved: 31.10.2014

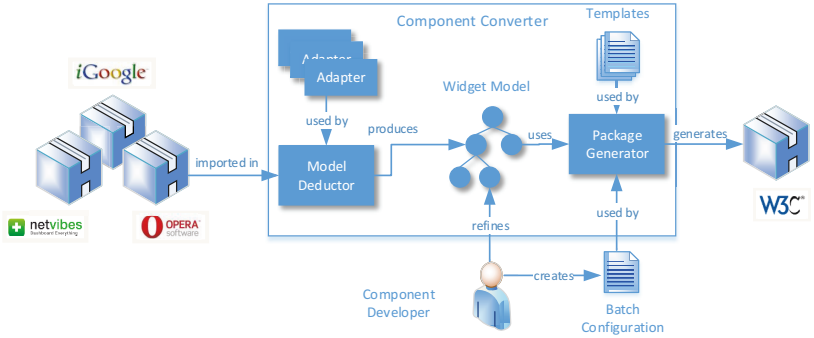


Figure 7.1.: Conversion of Proprietary Widgets into WebComposition/EUD Components

algorithm is applied to produce a WebComposition/EUD package out of the model (cf. Figure 7.1). In the following, the transformation steps are described in details.

Model Deduction

The goal of the first step is to derive a widget package model independent of a concrete packaging format. A corresponding metamodel that describes the abstract structure of WebComposition/EUD component packages (based on W3C Packaged Web Apps) is illustrated in Figure 7.2.

A WebComposition/EUD component package consists of 3 main ingredients: *Metadata*, *Configuration* and *Content Resources*. *Metadata* comprises a globally unique identifier and version of the widget as well as at least one localized set of further attributes. *Localized Metadata* consists of a name, a description and an icon of the widget. Additionally, it stores information about widget developer. *Configuration* data provides runtime parameters to the widget execution environment. Such param-

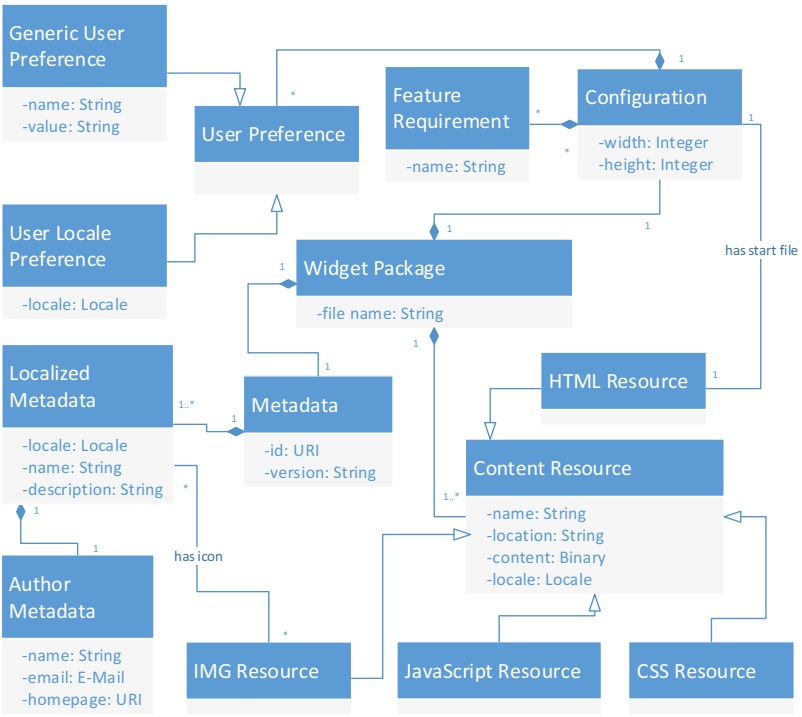


Figure 7.2.: Metamodel WebComposition/EUD Component Packages (based on W3C Packaged Web Apps)

eters are dimensions of the widget, a reference to the start file, so called *Feature Requirements* and *User Preferences*. *Features* are pluggable and configurable functionalities of widget execution platforms. An example of a *Feature* is some form of middleware that can be used by widgets to exchange data. A *Feature Requirement* states that a particular platform functionality is required for proper widget execution. In general, a widget cannot be instantiated if one of its *Feature Requirements* is not fulfilled. *User Preferences* that are also part of configuration data, contain default locale preference and generic name-value pairs. The latter can be used as initial parameters for business logic of a widget.

The last ingredient of a widget package – *Content Resources* – define business logic and user interface of a widget. Resources are optionally localized, i.e. there is a concrete locale assigned to each of the resources. Resources with specified locale provide language and region-specific translations of user interface or business logic. Runtime environments usually enable users to select a locale they prefer most. Each resource has a name, a relative location within the package and a content. The metamodel distinguishes between HTML, image, style sheet and JavaScript resources.

Model deduction takes place similarly for all source package formats. First, metadata is extracted from a given widget package. In case, identifier of a widget package or version are missing, the model attributes are set to automatically generated values. Referenced resources are downloaded and corresponding model elements are instantiated. If localization resources are available, the metadata is completed with corresponding entries. Then, configuration data is extracted. Dependencies identified in this step influence later adaptation of widget's source code. After that content of the original widget package is analyzed and new model elements (HTML, image, CSS and JavaScript resources) are created. Special considerations are required by HTML and JavaScript resources – the original widgets can rely on platform APIs that may be missing in other execution environments. For this purpose, package format-specific resources are added to models and corresponding HTML files are adjusted to include these resources.

In the following, some details on package model deduction for different widget formats are presented.

Opera Widgets The Opera widget specification[4] has been published 2007 and was based on the early W3C widget packaging draft. An opera widget is a compressed file with configuration, localization and content resources inside. The deduction algorithm extracts metadata and configuration files from a dedicated manifest file. Content resources are slightly modified to become compatible with current CSS standards and JavaScript APIs.

iGoogle Gadgets iGoogle gadgets have been used from 2005 till 2013 for personalized starting pages. An iGoogle gadget package is made of a single XML file that comprises gadget metadata, configuration and content resources (some resources are only referenced using Uniform Resource Locators (URLs) and, thus, have to be downloaded first). The model deduction algorithm has to take several peculiarities into account. For gadgets without inplace-content (of special type *url*) new HTML resources are created containing a single *iframe* element pointing to the given URL. New localization resources have to be generated based on so called message bundles referenced by a concrete package. As in case of Opera Widgets, new JavaScript resources (originating from related Apache Shindig[5] project) are added to emulate iGoogle-specific JavaScript APIs. Configuration of user preferences is implemented by extracting parameter names and their values from the original gadget package and by inserting preference management code into HTML resources. Authentication data is ignored as current W3C specification doesn't foresee corresponding attributes.

UWA Widgets The format has been introduced by Netvibes[6] for building personalized dashboards. Similar to the iGoogle gadgets, UWA

[4] `https://github.com/operasoftware/devopera-static-backup/blob/master/http/dev.opera.com/articles/view/opera-widgets-specification-11-fourth-ed/index.html`, Retrieved 3.6.2015
[5] `http://shindig.apache.com`, Retrieved: 3.6.2015
[6] `http://netvibes.com`. Retrieved: 3.6.2015

packages consist of a single XML file that comprises metadata, user preferences, markup and JavaScript code. Referenced resources are downloaded first. At the time of writing UWA specification didn't define any standard way of persisting localized resources or message bundles i.e., this aspect of UWA widgets is ignored. Furthermore, the format doesn't foresee definition of widget dimensions by widget developers - the resulting model should be completed in the next conversion phase.

If the automatically derived widget model misses some mandatory elements (e.g., configuration parameters or metadata), they have to be added to the model manually.

Generation of Component Packages

Generation of WebComposition/EUD component packages takes place automatically based on the model produced during Model Deduction step. The generation process can be divided into three steps: generation of the W3C Configuration Document; creation of folders with content resources and, finally, packaging of the content into one archive. The Configuration Document contains complete widget and author metadata (also localized one) and widget configuration including feature requirements and user preferences. Folders are created based on both the W3C specification (for locale-specific resources) and on relative paths of content resources. All files are then put into one *zip* archive named according to the original file name and date of conversion.

The algorithm has been implemented as a stand-alone tool with GUI interface. It enables both transformation of single packages and batch processing. If some information is missing, the information is completed either manually or using pre-configured values in a full-automatic mode.

7.2.4 Related Work

Automatic creation of UI components is widely applied for RSS (Winer, 2003) and Atom feeds (Nottingham and Sayre, 2005). Pre-defined templates are instantiated with concrete URLs and usually provide view, search, bookmark and sharing functionalities. The idea is applied by many dashboard platforms such as iGoogle, Netvibes and myYahoo[7]. However, the approach is rather limited as it produces components with very specific functionality only. Furthermore, it requires content providers to stick to the syndication formats supported by widget generation tools.

Several prototypes exist that produce W3C widgets based on existing code. One of them, so-called AppCache to Widget Converter (ac2wgt[8]), utilizes HTML5 Application Cache (Hickson et al., 2014) to enable access to HTML, CSS, JavaScript and media resources even if network connection is unavailable. The converter downloads all cachable files from a given resource, creates a configuration file and produces a W3C package file. The challenge is, however, to find sufficient number of input Web applications that are supported by the converter tool. Furthermore, at time of writing the software wasn't available for download.

Another prototype called *crx2widget*[9] transforms Google Chrome applications[10] into W3C packaged Web apps. The converter makes use of the fact that the structure of Chrome-native **.crx* files is very similar to the one specified by W3C specification and, therefore, they can be easily mapped on each other. The prototype has been successfully applied to "a few apps" from Google Web Store. It could be used in conjunction with the proposed tool, which follows similar idea but covers other input formats.

[7] `https://my.yahoo.com/`, Retrieved: 3.6.2015
[8] `http://berjon.com/hacks/ac2wgt`, Retrieved: 4.9.2012
[9] `http://scottbw.wordpress.com/2011/02/17/converting-chrome-installed-web-apps-into-w3c-widgets`, Retrieved: 3.6.2015
[10] `https://chrome.google.com/webstore`, Retrieved: 3.6.2015

7.2.5 Evaluation

In the following the tool is evaluated based on requirements stated in Section 7.2.2.

Compliance The package model used for generation algorithm has been derived from the current W3C Packaged Web App specification, which builds the basis of WebComposition/EUD components. The output of the algorithm is a W3C package file that corresponds to the specification and can be deployed to the component container of the composition platform. The requirement is considered to be fulfilled.

Efficiency Time and effort required for library population is minimal as widget transformation happens mostly automatically. Only if some data is not found in the original package, manual input is required. The batch processing mode enables to completely automate the process. However, as shown in the experiment below, the tool fails to convert some components correctly. In this case the broken packages have to be reviewed and repaired manually. Therefore, the requirement is considered to be partially fulfilled.

Extensibility The tool provides adapters to convert components of 3 different formats. The extensibility of the tool is given by pluggable adapter architecture. To integrate new component format, a dedicated adapter should be implemented that would produce the model defined in Section 7.2.3. The module for model refinement and generation of component packages are independent from source format. The requirement is considered to be fulfilled.

The success rate of the tool has been tested during an experiment with 25 most popular components[11] from each supported format (iGoogle, Opera and UWA). The components were downloaded from respective repositories, converted by the tool and then deployed to a widget container (Apache Wookie v0.11). Instantiated components were reviewed

[11] According to widget ratings on corresponding platforms on 13.07.2012

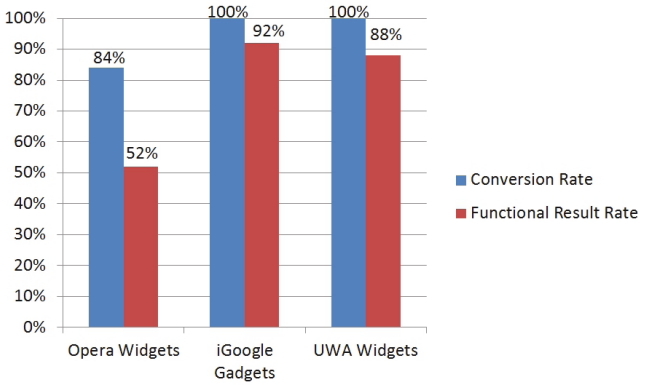


Figure 7.3.: Success Rate of WebComposition/EUD-CC

manually and their functionality compared with original widgets. The results of the experiment are presented in the Figure 7.3.

21 of 25 Opera widgets have been converted successfully. Two widgets used proprietary JavaScript APIs that were not supported by the tool. Two further widgets were invalid according to the Opera specification, although they were runnable in original Opera environment. Out of the 21 converted widgets only 13 functioned properly in Apache Wookie. The problems occurred mainly in Opera-specific API calls such as access to local hard drive files. The success rate for Opera widgets is therefore 52%.

All 25 iGoogle gadgets were converted successfully. 23 of 25 converted widgets worked correctly in Apache Wookie. One widget could not be imported because of a Wookie-related bug (already reported to the developer team at the moment of evaluation). Another widget caused several JavaScript exceptions after instantiation. The success rate of iGoogle gadgets is 92%.

The 25 selected UWA widgets were all converted into the W3C format. 22 of them worked correctly in Apache Wookie. 3 other widgets were in-

compatible with Wookie-provided proxy object and didn't work properly. The success rate for UWA widgets is, thus, 88%.

The overall success rate calculated over the three different widget formats is 77%. The findings from the above experiment show that automatic conversion of existing widgets is possible and feasible. Initial effort to implement adapters for specific component format is large. However, it pays off quickly if many source packages have to be converted into the WebComposition/EUD component format.

7.3 WebComposition/EUD ICCI Extender

A rich set of components enables construction of DSSs to support large number of different scenarios. With evolving requirements, however, Composition Developers might demand new components and composition possibilities to be provided by the platform. While completely new functionalities can be realized by adding new components to the repository, composition possibilities are often restricted by ICCIs of components. In WebComposition/EUD development process Component Communication Experts take care of the availability and quality of these interfaces. One of the most time-consuming activities is enabling newly added components to communicate with existing ones. Their source code has to be learned and manually extended with new publications and subscriptions, which is in general a time-consuming and error-prone task. If components have been imported from external repositories, support from original Component Developers is not guaranteed at all. Thus, it is desirable to provide support for Component Communication Experts, so that they can perform this task efficiently without the need to contact original Component Developers.

7.3.1 Motivation Scenario

A Component Repository has been populated with numerous components that have been developed manually by different developers or converted automatically using WebComposition/EUD-CC. Peter, a platform user, wants to extend his application for analysis of flood consequences with two new components – the one called PhoneDB, which would query phone numbers of public institutions in a given city, and the other one called VoIPCall, which would enable VoIP calls to a given phone number. The two components stem from different developers and were not designed to work together. Peter notices that they do not synchronize data among each other. He has to extract found phone number from results table of the one component and manually copy it into the other. Peter issues a feature request to extend the two components towards ICC – however, it will take significant time until the source code of the components gets updated. It would be beneficial for Component Communication Experts, if they could extend capabilities of repository components on their own. To avoid the need for learning the source code of components, semi-automatic facilities for such extensions are desirable.

7.3.2 Requirements

The following requirements can be derived on the envisioned assistance mechanism.

Effectiveness The assistance mechanism should enable extension of components with new ICC capabilities. Components should produce and consume notification messages as specified by users.

Efficiency The goal of the tool is to make Component Communication Experts more efficient in their work. Time and effort required to extend components with new capabilities should be minimized.

Usability The assistance mechanism should be easy to use and easy to learn. Users should perceive it as a useful tool that they also prefer to use.

The following section presents an interactive visual environment that addresses the above requirements.

7.3.3 Semi-automatic ICCI Extension

The proposed assistance mechanism enables interactive semi-automatic extension of component ICCIs. Publications and subscriptions can be mapped on state changes of component's GUI being represented by a DOM tree (cf. Section 4.3.1). Specification takes place using the PBD technique (Lieberman, 2001): users interact with the GUI of an instantiated component, while the tool records and analyzes performed user actions. If one of built-in action sequence pattern is detected, the corresponding actions can be further configured either as triggers for some outgoing notifications or as reactions (actions to be performed) to some incoming notification. The outgoing or expected notification messages are parametrized in a dedicated dialog based on values of elements involved into the detected actions sequence. Additionally to the PBD technique, the tool enables visual annotations of table structures. The goal is to make table data interactive by passing displayed values (cells or complete rows) as notification messages to other components. Extraction and publication take place upon cell or row selections.

The produced configuration is used by WebComposition/EUD-IE to automatically enrich component source code. After extension, component publishes and subscribes for notification messages as specified during demonstration or annotation. In the following details of the specification process and of automatic source code extension are presented.

Terminology

To describe the proposed solution we first introduce several assisting definitions. The concepts *user input action* and *pattern for user input action* are reused from Section 6.4.3. The set of element types in definition 6.4.1 is extended towards $T_e = \{TEXT - INPUT, MULTIPLE - CHOICE, BUTTON, TABLE - ROW, TABLE - CELL\}$. Additionally, the following definitions are introduced:

Definition 7.3.1. Let $A = \{a_i : a_i = \langle e_i, t_{a,i}, v_i \rangle, i = 1..n\}$ be a sequence of user actions performed in some component c. A pattern sequence $P = \{p_i : p_i = \langle e_i, t_{a,i} \rangle, i = 1..n\}$ that matches the sequence A is called *trigger-reactor pattern sequence*.

Definition 7.3.2. A *GUI-based ICC publication* pub_{GUI} is a tuple $\langle P_{PUB}, t_{PUB}, f_{PUB} \rangle$ with

- P_{PUB} being a trigger-reactor pattern sequence,
- t_{PUB} being some publish-subscribe topic,
- $f_{PUB} : P_{PUB} \longrightarrow string$ being a function that assigns string names to the patterns in P_{PUB}. The names should be later used as keys in published notification messages. For simplicity of implementation the keys should be unique: $\forall p_i, p_j \in P_{PUB}, i \neq j : f_{PUB}(p_i) \neq f_{PUB}(p_j)$

pub_{GUI} is a specification of component publication behavior. The pattern sequence P_{PUB} is used to identify user actions that should cause a new notification message to be published. t_{PUB} specifies the topic to be used and f_{PUB} the content of notification message.

Definition 7.3.3. A *GUI-based ICC subscription* sub_{GUI} is a tuple $\langle P_{SUB}, t_{SUB}, f_{SUB} \rangle$ with

- P_{SUB} being a trigger-reactor pattern sequence,
- t_{SUB} being some publish-subscribe topic,

- $f_{SUB} : P_{SUB} \longrightarrow string$ being a function that assigns string names to the patterns in P_{SUB}. The names are later used to find values in incoming notification messages to be assigned to elements from the pattern sequence P_{SUB}.

sub_{GUI} is a specification of component subscription behavior. The pattern sequence P_{SUB} provides a template for user input actions that should be executed on GUI of component c as soon as some notification message arrives.

Specification of ICC behavior

Specification of the required ICC behavior is performed using the PBD technique. Hereby a dedicated algorithm analyzes input actions made within a component and produces patterns for ICC publications or subscriptions (based on the choice of the operator). The topic to be used for notification messages and the naming function are provided manually. The two are of particular importance as they define the ICCI of the component. It is a developer responsibility to choose the topic and message attributes consistently in order to maximize semantic compatibility of extended component with others.

Algorithm 7.1 shows utilized rules to produce a trigger-reactor pattern sequence out of a buffered sequence of user input actions. On each new action the algorithm is executed anew. On each derived specification the buffer is reset.

The algorithm considers only action sequences that affect at least one input field and finish with button clicks. To illustrate the deduction process, consider the VoIPCall component from the motivation scenario above. To establish a call one has to type the phone number of callee into a dedicated input field and confirm the input using a *Call* button (cf. Figure 7.4).

Algorithm 7.1: Deduction of a trigger-reactor pattern sequence based on a given user input action sequence

Input : Component c, user action sequence $A = \{a_i = \langle e_i, t_{a,i}, v_i \rangle\}, i = 1..n$

Output : A trigger-reactor pattern sequence or $NULL$ if no deduction possible

1. If $e_n.t \neq BUTTON \vee \nexists i \in 1..n-1 : e_i.t \in \{TEXT-INPUT, DROPDOWN\}$ return $NULL$.
2. Otherwise, return $P = \{p_i : p_i = \langle e_i, t_{a,i} \rangle\}, i = 1..n$.

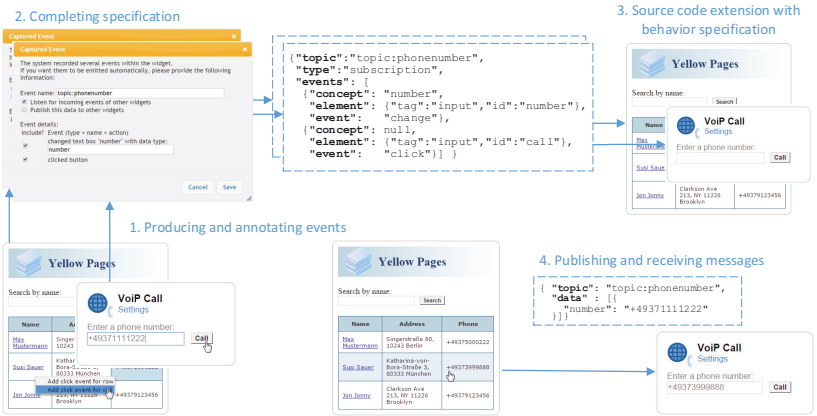


Figure 7.4.: Interactive Demonstration of Desired ICC Functionality

After each action, the algorithm 7.1 is executed. A click on the *Call* button produces a non-empty pattern sequence $P = \{p_1 = \langle input_{Phonenumber}, change\rangle, p_2 = \langle button_{Call}, click\rangle\}$. The topic to be used and the naming function should be provided by a user, e.g., $t =$ "$topic : phonenumber$" and $f = \{(p_1, "number")\}$. User can decide if the produced tuple $\langle P, t, f\rangle$ should cause a publication of entered phone number or if it should yield actions to be executed on subscriptions. For the above scenario it makes sense to make the component become a receiver of notification messages and not a sender.

Once GUI-based publications and subscriptions for a component were specified, it starts participating in ICC upon instantiation. The new *publication* behavior of a component is defined by the set of all specifications $\{pub_{GUI,i} = \langle P_{PUB,i}, t_{PUB,i}, f_{PUB,i}\rangle\}$. Any time user interacts with the component, his actions $A = \{a_k\}$ are recorded in a buffer and analyzed using the Algorithm 7.1. If the algorithm derives some trigger-reactor pattern sequence $P*$, it is matched against all of the pattern sequences $pub_{GUI,i}.P_i$. For every matched specification $pub_{GUI,j} = \langle P_{PUB,j}, t_{PUB,j}, f_{PUB,j}\rangle$, a notification message $m_j = \langle c, t_{PUB,j}, data_j\rangle$ is published, with $data$ being defined as follows: $data = \bigcup_{p_m \in P_{PUB,j} \wedge f_{PUB,j}(p_m) \neq EMPTY} (f_{PUB,j}(p_m), a_m.v_m)$. In other words, the notification message consists of values from the user input sequence A named according to $f_{PUB,j}$.

Subscription specifications $\{sub_{GUI,i} = \langle P_{SUB,i}, t_{SUB,i}, f_{SUB,i}\rangle\}$ make component accept notification messages on topics $t_{SUB,i}$. If a message $m = \langle c, t_{SUB,i}, data\rangle$ arrives, the following user action sequence A is executed on the GUI of the component: $A = \{a_j : a_j = \langle p_j.e, p_j.t, v\rangle, \forall p_j \in P_{SUB,i}\}$ where v is the value that corresponds to the key $f_{SUB}(p_j)$ in $data$ if such a key exists and empty string otherwise.

Architecture

The conceptual architecture of the WebComposition/EUD-IE is presented in Figure 7.5.

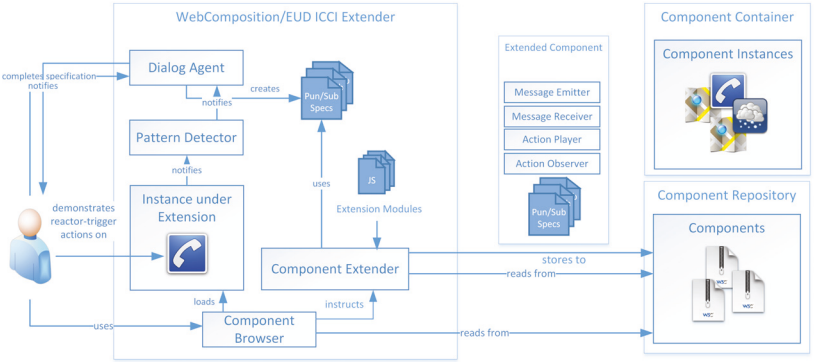


Figure 7.5.: Architecture of the WebComposition/EUD-IE

Using a dedicated browser user selects a component to be extended from Component Repository. Component Extender module then enriches the source code of the selected component with several functionalities that are able to observe and replay user actions on the GUI level. The included functionalities are:

Action Observer The module observes user-component interactions and notifies the environment about user input actions described by Definition 6.4.1. The module is used both during the specification of desired ICC behavior and during actual usage.

Action Player The module is able to replay a given sequence of user input actions. It is used to realize reaction routines on incoming notification messages according to provided specification.

Message Emitter If interactions observed by Action Observer match any pattern from given ICC specification, message emitter constructs corresponding notification messages and publishes them on specified topic.

Message Receiver The module receives incoming notification messages and constructs action sequences that should be replayed by Action Player.

After required modules have been added to component source code, the component is instantiated and the instance is loaded into the WebComposition/EUD-IE. A user then demonstrates input actions that should act as triggers or reactors to notification messages, and annotates table elements to extract the data from. The so-called *Pattern Detector* applies the Algorithm 7.1 to derive pattern sequences and passes them to the *Dialog Agent*, which asks the user for topic and naming function to be used. The resulting specification is then automatically added to the source code of the component and new package is stored in the repository. From then on, component can be instantiated and used in the composition environment in conjunction with other components.

The tool is implemented as a stand-alone ASP.NET Model-View-Controller (MVC) application. Apache Wookie, part of the Composition Platform acts as Component Repository and Container. The extension modules are JavaScript libraries that use OpenAjaxHub for communication with the main application. Recorded ICC specifications are stored as JSON-objects. Additionally, configuration documents of W3C widgets are extended with descriptions of component ICC capabilities.

7.3.4 Related Work

A few projects have tackled the problem of component extension towards ICC so far. The scrapplet.com[12] mashup platform enables development of mashups out of UI components. Mashup developers can extend aggregated components with publish-subscribe messaging capabilities by adding snippets of Javascript that are executed on GUI events such as clicks, key presses etc. Similarly, callback routines can be defined to handle incoming messages. The tool supports developers in injection of communication libraries, with code templates and with GUI-based

[12] `http://scrapplet.com`, Retrieved: 5.6.2015

topic definition. However, the Javascript code has to written manually and there is no support for dealing with long sequences of input actions. Scrapplet offers higher flexibility than the WebComposition/EUD-IE, but requires stronger Javascript skills and is more time-consuming.

In (Daniel and Matera, 2009) authors propose to convert traditional Web applications into mashup UI components by means of source code annotations. A dedicated wrapper loads then applications into containers that equip them with ICC functionality. The wrapper utilizes a dedicated microformat within application's HTML code and a separate component description document. The microformat enables annotation of application UI controls with information on associated application-level events and parameters. Although the approach has higher flexibility than WebComposition/EUD-IE, it still requires manual and, thus, time-consuming source code modification.

7.3.5 Evaluation

The Efficiency and Usability requirements have been evaluated within a dedicated user study. The goal of the evaluation was to test, if the proposed tool speed-ups the ICCI extension process, and to assess its usability. To answer these questions a laboratory experiment with 10 participants has been conducted and its results have been evaluated using statistical hypothesis testing (two-tailed t-test with significance level 95%). The following two null hypotheses were tested:

H1. For given scenario the average time to perform an ICCI extension using the tool is lower than the average time for the same task but without the tool.

H2. For given scenario and time constraints the success rate in making extended components loop-resistant is higher if ICC extender is used.

Usability properties of the tool have been assessed with average user ratings based on a 5-point Likert scale. Complete data collected during evaluation is given in Appendix B.4.

Setup

Overall 10 participants (students, PhD students and researchers of the Faculty of Computer Science, Technische Universität Chemnitz, Germany) took part in the evaluation. All the participants had basic or advanced Javascript skills. 90% of them have already used or developed Web-based UI components. However, only 20% of the participants had prior experience in development of W3C widgets. ICC was a familiar concept for everyone - 50% have seen it working and 50% developed ICC behavior on their own. OpenAjaxHub was a new concept to 40% of the participants.

WebComposition/EUD-IE has been installed locally on three workstations with similar hardware and software characteristics. Local installations of Apache Rave 0.17 and Wookie 0.11 were used. All participants used Notepad++[13] for manual source code modifications.

Procedure

The evaluation procedure took place as follows. First, the participants filled a questionnaire, where they assessed their skills and experience in development of Web-based components. Then they were given an introduction into the WebComposition/EUD component model and underlying technologies (W3C packaging format and OpenAjaxHub API). One example for extending two components manually and using the WebComposition/EUD-IE was shown. After that each participant had to solve two tasks related to extension of components towards ICC – once without the tool and once with. The time required for

[13] `http://notepad-plus-plus.org`, Retrieved: 5.6.2015

completion of the first task has been measured. Regarding the second task it was only checked, if the task was completed in a fixed time span. Finally, the participants filled a post-evaluation questionnaire, where they assessed the usability, understandability, applicability and reusability of the tool.

In the first task participants were asked to extend three given components towards ICC (cf. Figure 7.6). One component called *MovieList* enabled browsing through a collection of movies and search by keywords. The second one called *WikiSearch* showed Wikipedia[14] articles related to an entered concept. The last one called *YoutubeBrowser* searched for Youtube[15] videos using a given term. Originally the three components do not share any data to others. The goal of the first task was to extend their source code, so that a keyword entered into the *MovieList* component was automatically passed to the two others upon its submission. The other components had to show their respective search results to the passed keyword. The second task required a similar behavior but into another direction: if a keyword had been entered into the *WikiSearch* component, it should have been passed to the *YoutubeBrowser* and *MovielList* ones. The complexity of the task lied in prevention of message loops that would occur between *MovielList* and *WikiSearch* as soon as one of them publishes a notification message that can be consumed by the other. Depending on the implementation, a receiving component can retransmit the incoming keyword and, thus, cause a self-reinforcing message exchange.

Results

All participants were successful in the first task (both manually and using the tool) and extended components towards ICC correctly. The time required for solution of task 1 using the tool was at least 50% shorter (cf. Figure 7.7), indicating that the tool makes developers more efficient. The hypothesis H1 is accepted.

[14] `http://en.wikipedia.org`, Retrieved: 5.6.2015
[15] `http://www.youtube.com`, Retrieved: 5.6.2015

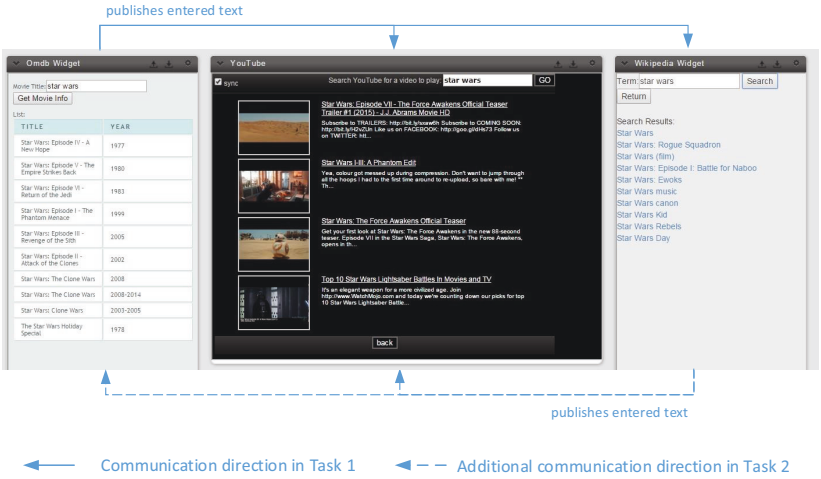


Figure 7.6.: WebComposition/EUD-IE: Evaluation tasks

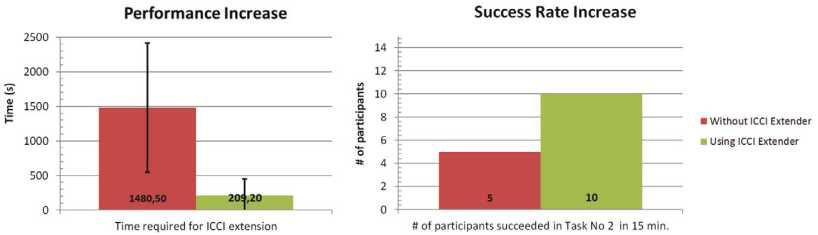


Figure 7.7.: WebComposition/EUD-IE: Performance and Success Rate Increase

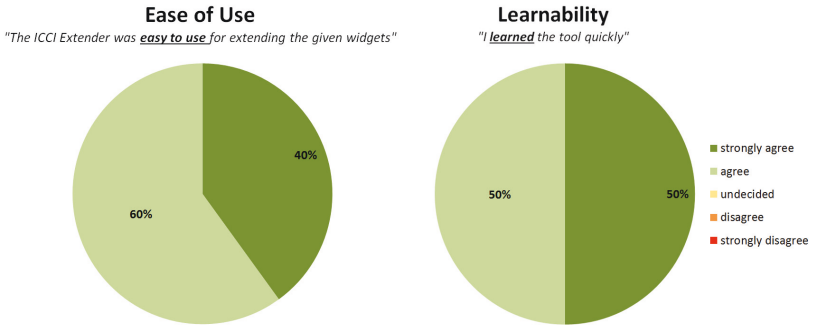


Figure 7.8.: WebComposition/EUD-IE: Ease of Use and Learnability

The loop prevention task (manual try) has been solved by 50% of participants only (in given time constraint). In contrast, all the participants completed the task in time using the tool. The hypothesis H2 is accepted. The results suggest that the tool makes Component Communication Experts more efficient in ICCI extension tasks.

The post-questionnaire analysis yielded the following results. 70% of participants agreed or strongly agreed that the manual extension process had been cumbersome, which once again underlines the necessity for assistance mechanisms. Other ratings indicate that the tool was easy to use (60% agreed and 40% strongly agreed) and easy to learn (50% agreed and 50% strongly agreed) (cf. Figure 7.8).

70% of the participants found the WebComposition/EUD-IE well applicable for extension of Web widgets in general (cf. Figure 7.9, left). However, 10% were undecided and 20% disagreed, which indicates that some participants were aware of situations that could not be handled by the tool. The reason surely lies in the fact that WebComposition/EUD-IE focuses on monitoring and analysis of UI state changes only (form inputs, form submissions and static tables). As indicated in recommendations by participants, further event specification methods should be explored to improve applicability of the tool.

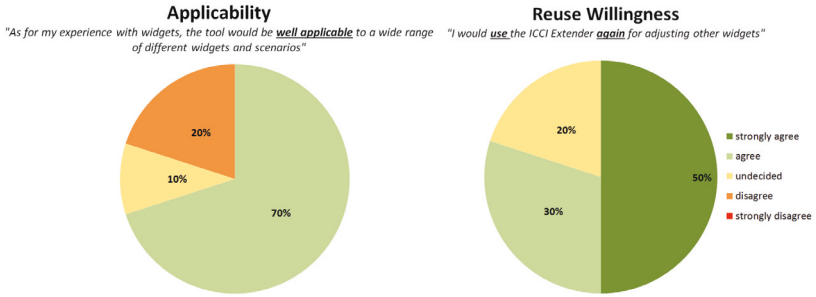


Figure 7.9.: WebComposition/EUD-IE: Applicability and Reuse Willingness

80% of the participants agreed or strongly agreed that they would reuse the tool in the future. 20% were undecided.

The post-evaluation questionnaire delivered many suggestions on how to improve the tool and which aspects of ICC extension process are especially important to support. For example, many participants suggested to automatically extract text from anchor elements or to react to events other than form submissions. Some missed a possibility to edit action sequences once they have been recorded – a well-known challenge in PBD-based systems. Also, a better control of when recording starts and ends was desired. Some participants complained about ambiguous explanations given by the tool. One participant suggested improving the quality of automatically extracted data by e.g., focusing on popular Javascript MVC frameworks. Incorporation of the above comments should be taken into account in future environments for semi-automatic extension of ICCIs.

The participants also described, which activities of the extension process were the most cumbersome ones and which of them they would like to be supported in. Understanding source code of widgets and finding elements for event binding have been mentioned several times. Also, loop prevention was considered as a non-trivial task. Finally, participants mentioned packaging and message compatibility making as cumbersome

activities to be supported. The feedback provides further improvement points for WebComposition/EUD-IE and new aspects to focus on.

In the following, the requirements from the Section 7.3.2 are reviewed.

Effectiveness The WebComposition/EUD-IE is able to extend components towards ICC based on monitoring and replay of events on the UI level. The extended widgets behave according to the communication model from Section 4.3.1. As also indicated in the study, the tool focuses on form- and table-based components and is only partially applicable to other types. Thus, the effectiveness requirement is considered to be partially fulfilled.

Efficiency The user study showed that the tool increases performance of Component Communication Experts. The success rate of preventing a looping behavior also increases. The requirement is considered to be fully fulfilled.

Usability Two aspects of the usability, namely learnability and ease of use, were confirmed by participants in the user study. Though there is an improvement potential in the explanations and messages shown by the tool, the requirement is considered to be fully fulfilled.

7.4 WebComposition/EUD Artifact Library

Maintenance of artifacts that are produced during WebComposition/EUD development process is a time-consuming activity and becomes even more challenging with growing set of deployed components and produced compositions. To retain their reusability and ease of discovery, metadata, documentation and history of WebComposition/EUD artifacts should be systematically organized and maintained. Dedicated tools are required that would help Artifact Library Managers, Component

Developers and Component Communication Experts to perform these activities time- and cost-efficiently.

7.4.1 Motivation Scenario

To illustrate the problem consider the following scenario. Alex, an Artifact Library Manager, collects and stores artifacts produced by Component Developers, Component Communication Experts and Composition Developers in different repositories. Components and Compositions are stored in respective repositories, documentation and feature requests – in company-internal file server, topic definitions and respective schemes – in a restricted Web space. If Alex wants to update or add a new component, he needs to find and update data distributed in different locations, which is a time-consuming task. Sharing and access to resources (e.g., installed components, documentations or problematic composition models) is inefficient due to heterogeneity of access protocols, data formats and access control mechanisms.

A solution that would centralize and support storage, management and sharing of different types of resources, could significantly optimize platform maintenance processes.

7.4.2 Requirements

Based on the above considerations, the following requirements can be derived.

Heterogeneous Content Storage and management of composition models, component packages, various metadata and documentation should be supported.

Access Control Artifact Library Managers should be able to define fine-grained rules for access and modification of stored resources.

Heterogeneous Interface Integration of the solution into third-party systems should be facilitated by means of flexible and configurable API.

In the following a dedicated repository for WebComposition/EUD artifacts is presented.

7.4.3 Artifact Access and Management

The proposed solution is based on Data Grid Service (DGS) – an extensible and configurable Web service that provides uniform access to heterogeneous resources (Chudnovskyy, Wild, et al., 2012). Each resource type is handled by a dedicated plugin called *Data Space Engine* (cf. Figure 7.10). DGS takes care of plugin configuration, metadata management, routing of incoming client requests and access control.

Clients communicate with DGS over the uniform REST/HTTP interface. The generic access control layer checks if a requester is authorized to perform given action on the given resource. If so, the request is analyzed by the *Request Router* module and forwarded to the responsible Data Space Engine. All modules have access to the global metadata, which represents a knowledge base about available resources, their configurations and relationships. The response produced by Data Space Engines is directly forwarded to the requesting client.

To enable management of various WebComposition/EUD artifacts, the following Data Space Engines were integrated:

XML Data Space Engine The engine is used to manage so called XML lists – data structures, consisting of pieces of XML. Each item of the list can represent a single composition model in OMDL format, a structured feature request or a reusable definition of transformation function. The plugin provides data validation and search functionality.

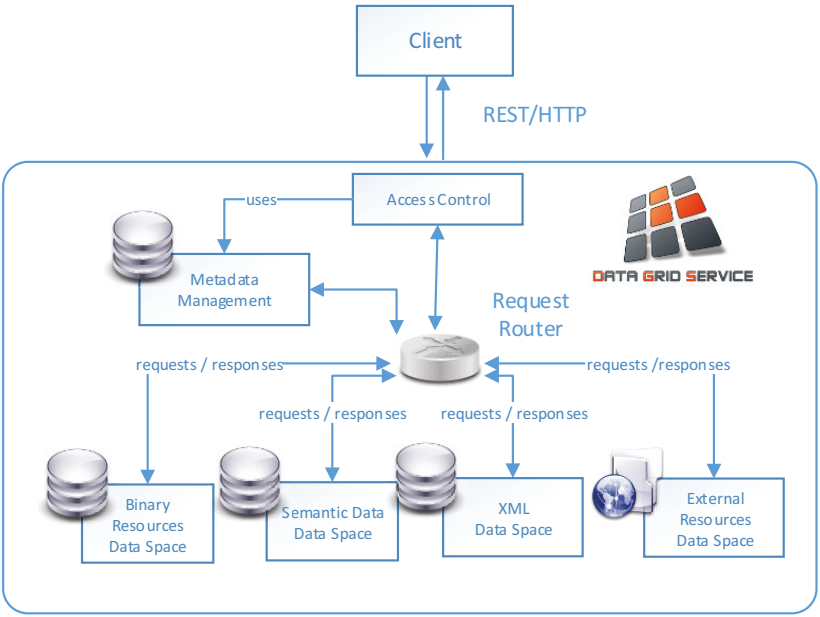


Figure 7.10.: Architecture of the WebComposition/EUD-AL

Binary Data Space Engine The engine provides Create, Read, Update, Delete operations for binary files with possibility of automatic publishing of their metadata in semantic formats. The metadata is extracted for several file formats such as W3C packaged Web apps, PDF, JPEG and MP3 resources. The engine can be used to store and access WebComposition/EUD components as well as accompanying documentation.

Semantic Data Space Engine The engine manages resources created in RDF format and enables their transformation in alternative representations. The engine is used to store and to provide access to metadata of WebComposition/EUD artifacts.

In the following, details of the integrated Data Space Engines are presented.

XML Data Space Engine

The main purpose of the XML Data Space Engine is to manage so called XML lists i.e., XML resources that have a flat tree structure and contain a list of semantically and structurally related XML items. XML Data Space Engine is implemented on top of BaseX 7.6[16] - an open-source XML database. By restricting the view from general XML resources to XML lists, the Data Space Engine can provide additional functionality specific to lists. In particular, the module provides operations to identify, retrieve, search for single items, append new and delete existing ones. Each item is uniquely identified by its primary key - an attribute or sub-element that is computed and filled automatically by the engine. Primary keys can be defined for each list separately.

XML Schema is used to define a valid structure of the elements and to guarantee the list integrity. Each time the list is changed, it is validated against a pre-defined XML schema. Either list- or element-wise validation is possible. The post-processing of the XML data can be specified

[16] `http://basex.org`, Retrieved: 7.6.2015

using a XSL Transformations (XSLT) stylesheet. This way, arbitrary representations of original data can be produced.

Search within the dataset is performed using so called XQuery templates. Hereby, user associates a URL pattern with an XQuery command. The command contains placeholders that are filled with actual values from the URL at run-time (cf. Listing 7.1).

Listing 7.1: Definition of XQuery Templates

```
prefix meta:
<http://www.webcomposition.net/2008/02/dgs/meta/>.

<http://artifactslibrary.example.org/compositions>
meta:queryTemplate
[
meta:url "compositions/query:{value1}:{value2}";
meta:query "for␣$x␣in␣[list]/*
where␣$x[contains(description,'{value1}')]
order␣by␣$x/title/text()
return␣$x/{value2}"
].
```

The above pattern, which is defined in Turtle format (Beckett et al., 2014), makes the service to respond to URLs such as $http://artifactslibrary.example.org/compositions/sms/title$. The data space engine returns then an XML document with titles of those compositions that contain the keyword "sms" in their description.

Binary Data Space Engine

The Binary Data Space Engine implements Create, Read, Update, Delete (CRUD) functionality for binary resources. The latter are grouped in collections and are exposed to service clients as Atom feeds (Nottingham and Sayre, 2005). Standard HTTP methods are used to add, update or delete the resources. Annotations and metadata is exposed in

RDF-format and can be retrieved from a well-defined URL - {*resource URL*}*/meta*.

Automatic metadata extraction is defined for several resource types – W3C packaged Web apps, PDF, JPEG and MP3 formats. Format-specific metadata is mapped onto well-known RDF vocabularies such as Dublin Core Metadata Terms [17], Music Ontology[18] and EXIF ontology[19]. An example of the automatically extracted W3C Packaged Web App metadata is given in Listing 7.2.

The thumbnail of the widget is extracted and stored as a separate resource. Listing 7.3 shows EXIF metadata of the JPEG-thumbnail.

Semantic Data Space

Semantic Data Space Engine provides configurable access to RDF metadata of the service and its resources. Its SPARQL endpoint enables arbitrary queries to the RDF dataset stored in a Sesame triple store[20], whereas full text search is facilitated by internal indexing provided by uSeekM library[21]. The engine enables definition of so called SPARQL templates – rules to map URL patterns onto SPARQL queries. The result can be transformed depending on the content type of the incoming request. The transformation can be configured using an XSLT stylesheet. Figure 7.11 shows an example of the template and its processing.

An incoming URL is matched against URL patterns of stored SPARQL templates. The first match is activated, meaning that actual parameters from the incoming URL are inserted into the SPARQL pattern and the resulting query is executed against the service metadata store. The result is transformed using the corresponding transformation algorithm.

[17] `http://dublincore.org/documents/dcmi-terms/`, Retrieved: 7.6.2015
[18] `http://musicontology.com`, Retrieved: 7.6.2015
[19] `http://www.kanzaki.com/ns/exif`, Retrieved: 7.6.2015
[20] `http://rdf4j.org`, Retrieved: 7.6.2015
[21] `https://dev.opensahara.com/projects/useekm`, Retrieved: 7.6.2015

Listing 7.2: Automatic Extraction and Publishing of Packaged Web App Metadata

```
<rdf:Description
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:wdg="http://www.w3.org/ns/widgets/"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:om="http://www.ict-omelette.eu/schema.rdf#"
     rdf:about="https://artifactslibrary.example.org/
binaries/testwidget.wgt" >

<rdf:type
     rdf:resource="http://www.ict-omelette.eu/schema.rdf#Widget" />
<dc:title>HelloWorldWidget</dc:title>
<dc:description>This is my first widget</dc:description>
<dc:creator>Max Mustermann</dc:creator>
<dc:language>en-us</dc:language>
<wdg:height>300</wdg:height>
<wdg:width>400</wdg:width>
<wdg:viewport>windowed floating</wdg:viewport>
<wdg:icon rdf:resource="https://artifactslibrary.example.org/
binaries/b55-icon.jpg" />
<wdg:hasFeature>
        <rdf:Description>
                <rdf:type
                     rdf:resource="http://example.org/api/geolocation"
                     />
                <wdg:required>false</wdg:required>
                </rdf:Description>
</wdg:hasFeature>
<om:commercialLicense>This is my license</om:commercialLicense>
<wdg:version>1.0 Beta</wdg:version>
</rdf:Description>
```

Listing 7.3: Automatic Extraction and Publishing of Image Metadata

```
<rdf:RDF xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dct="http://purl.org/dc/terms/"
xmlns:exif="http://www.w3.org/2003/12/exif/ns#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="https://artifactslibrary.example.org/
binaries/b55-icon.jpg">
        <dct:created>13.05.2014 10:04:37</dct:created>
        <dct:creator
             rdf:resource="http://people.example.org/webids/max#me"
             />
    <exif:date>13.05.2014 10:04:37</exif:date>
    <exif:copyright>VSR-2014</exif:copyright>
    <dc:format>image/jpeg</dc:format>
    <exif:software>Adobe Photoshop CS6 (Windows)</exif:software>
    <exif:make>Canon</exif:make>
    <exif:model>Canon EOS 5D Mark II</exif:model>
  </rdf:Description>
</rdf:RDF>
```

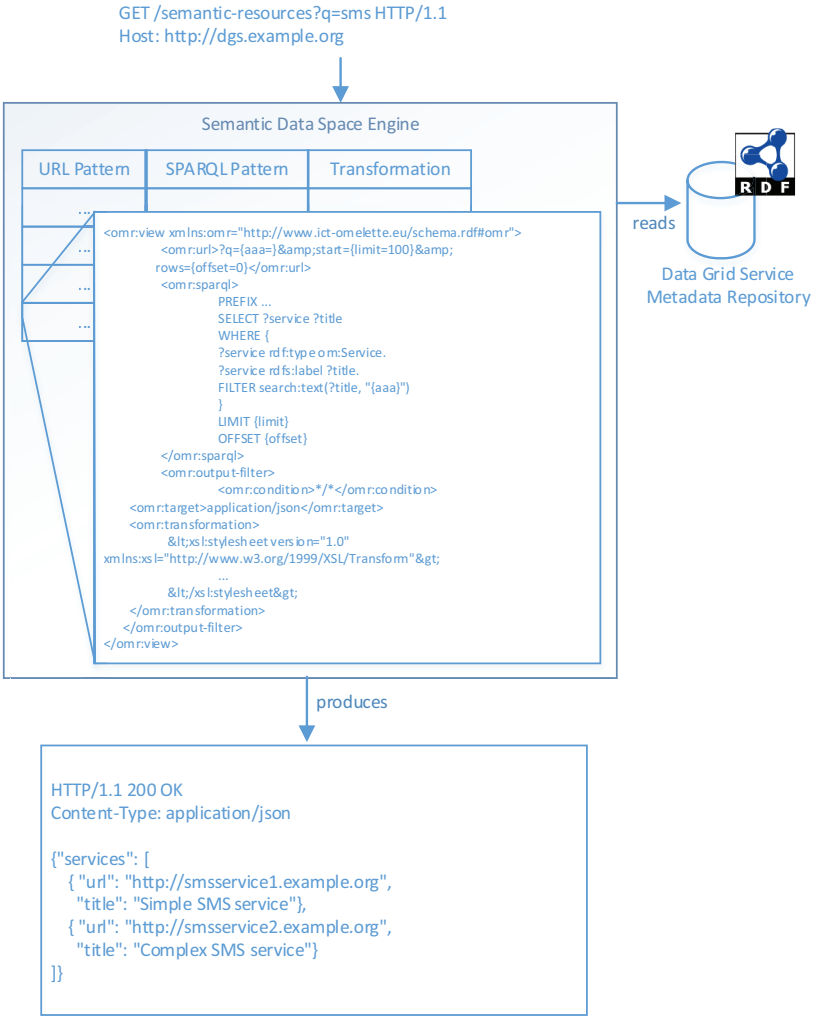


Figure 7.11.: Processing of SPARQL Templates

With the help of SPARQL templates, different representations of RDF resources can be produced. The EDUKAPP JSON API[22] has been implemented to enable integration of the DGS into Apache Rave.

Service Access Control

Access to service resources can be controlled both on the level of single service clients and on the level of groups identified by common properties or mutual relationships. Authorization is performed on the resource/operation basis. The DGS doesn't perform any identity management, but rather relies on self-managed identities using the WebID identification mechanism (Sambra et al., 2014).

WebID enables decentral persistence and management of digital identities. Identity attributes can be arbitrary claims that users (or machines) make about themselves, about other users or about mutual relationships. One important attribute (called WebID) is the IRI of a resource that provides RDF description of identity attributes (called WebID profile). Another important attribute is a cryptographic key pair for binding a WebID profile to the identity.

WebID authentication takes place during SSL/TLS handshake between a client (e.g., a user agent) and a server. A client claiming to possess a particular WebID identity presents a self-signed X.509v3 certificate that is logically and physically bound to the claimed identity as follows:

1. The certificate's public key is the public key of the identity
2. The certificate contains the identity's WebID in its *Subject Alternative Name (SAN)* extension field
3. The referenced WebID profile contains among others the identity's public key

[22] `https://code.google.com/p/edukapp/wiki/JsonApi`, Retrieved: 4.7.2015

To check the claimed identity possession, the server performs several steps. First, the presented certificate and its key are validated based on the SSL/TLS validation procedure. Second, the server reads WebID from the certificate and fetches the corresponding WebID profile. Finally, it checks if the profile contains a public key from the presented certificate. If all the steps succeed, the server assumes that the identity represented by the profile belongs to the client. It should be noted that due to self-managed and self-hosted identities, the above process doesn't check trustworthiness of identity claims others than write access to the WebID profile and possession of the keypair.

The authorization takes place using identity- and group-based access control lists. Authorization rules are specified using the Web Access Control (WAC) mechanism[23], which enables decentralized management of group memberships. An authorization rule is a triple consisting of a resource to be accessed, set of permissions (read, write, control, append) and an IRI being either a WebID or an identifier of a group. Permissions to access a resource are granted, if the WebID of an accessing client is explicitly listed in one of the corresponding authorization rules or if there is a group in those rules that contains the WebID.

Listing 7.4 shows a WAC access control list that enables read access to a topic specification to any client, but write access only to the Component Communication Expert with WebID *http://homepage.example.org/foaf#joe*.

Listing 7.4: Example of Access Control Rules Definition in Data Grid Service

```
@prefix acl: <http://www.w3.org/ns/auth/acl>.

[acl:accessTo <http://artifactslibrary.example.org/
    topics/location>; acl:mode acl:Read, acl:agentClass
    foaf:Agent].

[acl:accessTo <http://artifactslibrary.example.org/
    images/img.png>; acl:mode acl:Read, acl:Write; acl:
    agent <http://homepage.example.org/foaf#joe>]
```

[23] `http://www.w3.org/wiki/WebAccessControl`, Retrieved 7.6.2015

WAC enables four access permissions: *Read*, *Write*, *Append* and *Control*. *Read* permission enables read-only access to a resource. *Write* enables all kind of modifications including complete resource removal. *Append* is a permission to add content to the resource only. Finally, *Control* is used to enable modification of the associated access control list. For the REST/HTTP interface the operations have been mapped onto HTTP verbs as follows: *Read* maps on *GET*, *Write* on *POST*, *PUT*, *DELETE* and *Append* on *POST*. *Control* is ignored due to the fact that the access control list is maintained internally by Artifacts Library Manager.

7.4.4 Related Work

Several research project targeted the problem of efficient access and maintenance of heterogeneous data.

data.fm[24] is an open-source Web service that supports enables management of structured data over a RESTful API and GUI. Access to the data, which is organized in files and directories, can be restricted. Compared to DGS, data.fm doesn't focus on automatic metadata extraction and configurable resource representations.

OpenLink Virtuoso[25] is a service that supports management of heterogeneous data sources. It operates on different types of structured data and provides various APIs such as SOAP, REST/HTTP, XML-RPC, and SPARQL. However, automatic extraction of resource metadata is not supported.

iServe (Pedrinaci et al., 2010) is a research prototype for publishing Semantic Web Services as Linked Data. It makes use of semantic service descriptions and enables users to search for services based on their "high-level" APIs. The registry provides capability for free text search as well. iServe focuses on semantically described resources only and lacks support for other types such as widget packages or XML documents.

[24] `http://data.fm`, Retrieved: 7.6.2015
[25] `http://virtuoso.openlinksw.com`, Retrieved: 7.6.2015

7.4.5 Evaluation

In the following, the requirements stated in section 7.4.2 are evaluated.

Heterogeneous Content The proposed solution consists of several modules that are able to persist various kinds of content. As described above, structured content in form of XML, arbitrary binary content and semantic metadata are supported. Beside basic CRUD functionality the corresponding modules implement filtering and manipulation methods specific to the type of content they handle. As a result, the service can persist composition models in OMDL format, WebComposition/EUD components, their semantic metadata and various structured content such as topic definitions, feature requests etc. The requirement is considered to be fully fulfilled.

Heterogeneous Interface The default interface of the DGS is REST/HTTP. For reading operations the XML content can be transformed into other formats by means of XSLT stylesheets. Similarly, semantic content can be exposed in different formats using the SPARQL templates. Such interface flexibility enables the service to be integrated into third-party applications as it is done for Apache Rave. The requirement is considered to be fully fulfilled.

Access Control Access restrictions on resources within the DGS can be expressed using WAC lists. Restrictions can be defined both for single identities or for their groups. The rules are evaluated on each HTTP request. In case of SPARQL requests, however, the mechanism works only partially – one can either enable or forbid all SPARQL requests without any fine-grained differentiation. The requirement is therefore considered to be partially fulfilled.

Several performance measurements have been conducted to test applicability of the WebComposition/EUD-AL to real-life scenarios. For this purpose the DGS has been populated with 250000 component descrip-

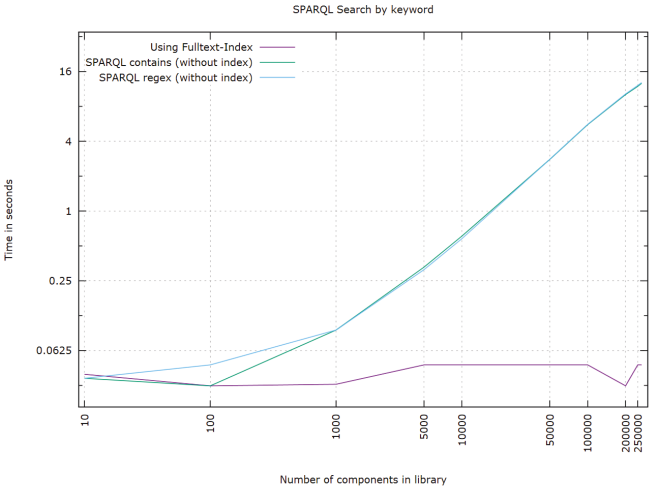


Figure 7.12.: Response Times of DGS for Component Search Queries by Keyword

tions from Netvibes repository[26]. The tests should check if the library was capable to handle given amounts of data and if the response time was below 2 seconds, which is considered to be an upper limit that doesn't impact user experience (Nah, 2004). The tests run on a local installation of DGS on a machine with X CPU, 4GB Memory, 120GB hard drive and Windows 7 as operating system. The library was populated in logarithmic steps, after which different read and write operations were issued. The time between requests and responses has been measured.

Figure 7.12 shows time measurements that were collected while searching for components by a pre-defined keyword. The results show that the uSeekM indexing engine significantly speeds up free text search operations. Standard SPARQL operations in contrast exceed the limit of 2 seconds already by 50000 components. Figure 7.13 shows results of search operation based on semantic reasoning. Components were annotated with property *rdf:type* and evenly distributed into two groups

[26] `http://eco.netvibes.com`, Retrieved: 7.6.2015

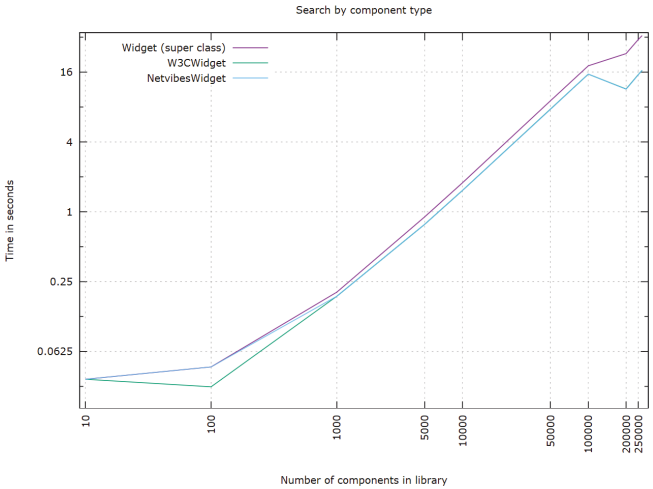


Figure 7.13.: Response Times of DGS for Component Search Queries by Component Type

W3CWidget and *NetvibesWidget*. The two concepts have been declared to be subclasses of *Widget*. Time measurements show that requests that involve the semantic reasoner are much more time-intensive. The limit is exceeded by approx. 20000 entries in the library.

Figure 7.14 shows time measurements for publishing and accessing new components. The results show that the values are almost independent of the number of overall components and do not exceed the limit of 2 seconds.

7.5 Summary

This chapter presented the last chain of WebComposition/EUD framework that consists of mechanisms to support evolution of WebComposition/EUD components and compositions. It started with description of

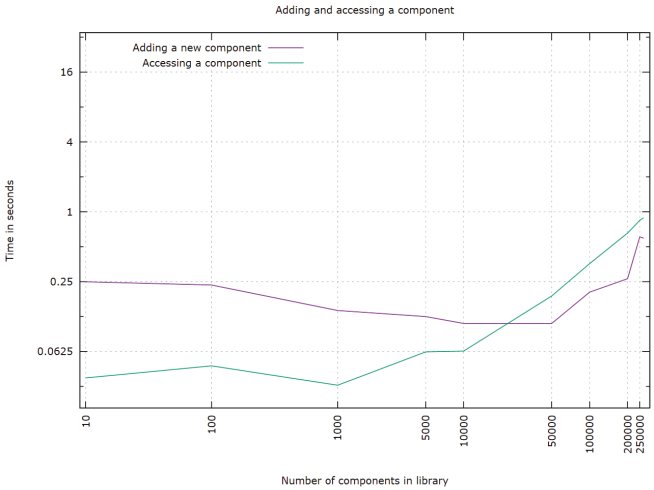


Figure 7.14.: Response Times of DGS for Adding and Accessing New Components

WebComposition/EUD Component Converter – a tool that helps Component Developers to develop and provide new functionalities by automatically converting proprietary Web-based widgets into WebComposition/EUD components. Then it presented WebComposition/EUD ICCI Extender being a tool that targets Component Communication Experts and helps them to efficiently improve composability of WebComposition/EUD components by enriching their ICCI interfaces. The chapter concluded with presentation of WebComposition/EUD-AL that supports Artifact Library Managers in maintaining platform artifacts and organizing access and their usage by interested parties.

Overall Evaluation 8

This chapter has a goal to evaluate the WebComposition/EUD approach according to the requirements stated in Chapter 2. Afterwards, its benefits and weaknesses compared to the state of the art are analyzed. Finally, the chapter describes several use cases and application domains, where the framework has been practically applied.

8.1 Requirements Evaluation

In the following the process model and the toolkit of the WebComposition/EUD approach are analyzed in the context of the respective requirements.

8.1.1 Development Process

The requirements on the development process are fulfilled to the following extent:

D1: End-User Involvement: The WebComposition/EUD development process enables non-programmers to perform development of composite DSSs autonomously. Given sufficient number of reusable building blocks, users can address many of problems related to

decision making by combining these blocks in appropriate way. Some activities in the WebComposition/EUD development process, however, are assigned to professionals as they require technical skills and programming experience that cannot be expected from non-programmers. These activities include development of new components, updates to existing ones and tasks related to management of produced artifacts. According to the assessment scheme, the requirement is partially fulfilled.

D2: Process Conciseness: The WebComposition/EUD process consists of two stages - the bootstrap and continuous evolution. The former, which doesn't involve end users, foresees all activities of systematic software development and prepares a dedicated infrastructure required for the subsequent phase. The latter, however, is reduced to three steps only, whereas the process finishes, if appropriate solution has been found during the Solution Discovery step. This approach specifically addresses needs and capabilities of non-programmers working under time constraints. The development process is therefore concise and the requirement is fully fulfilled.

D3: Reuse-Orientation: The proposed development process promotes production and reuse of software artifacts at several stages. In the bootstrap phase, Component Developers build WebComposition/EUD components – reusable building blocks for composition of DSSs. Component Developers are free to choose the development method, however supplied WebComposition/EUD tools promote reuse of existing services and visual components and their conversion towards WebComposition/EUD components. In the evolution phase, reuse represents the core of DSS construction activities. Reuse takes place on the level of solutions (Solution Discovery) and on the level of components (Customization). Produced compositions extend the library of reusable artifacts and contribute to the efficiency of future Composition Developers. The degree to which the process focuses on reuse of existing artifacts is high. The requirement is fully fulfilled.

8.1.2 Development Toolkit

The requirements on the tool assistance are fulfilled to the following extent:

T1: Development Assistance: Development of data-driven DSSs is enabled and facilitated by the WebComposition/EUD composition platform (cf. Chapter 5). The platform enables composition of data sources and functions if these are wrapped according to the WebComposition/EUD composition model. The freedom of data source / functionality selection offered by the platform is high. The logic that can be defined on top of aggregated components consists of restrictions, isolations and transformation functions. The expressiveness of the integration logic depends, however, on the ICCIs of components. Finally, freedom of configuration of user interface is average as platform provides a limited set of possibilities for layout customization by end users. The requirement is partially fulfilled.

T2: Evolution Assistance: The WebComposition/EUD toolkit provides three mechanisms that have a goal to make management and maintenance of user-produced solutions more efficient (cf. Chapter 7). New functionalities and data sources can be automatically produced by reusing code of proprietary components. Extensions to interfaces of WebComposition/EUD components (and thus ensuring interoperability of new and old ones) can be performed semi-automatically and in an interactive fashion. Finally, produced software artifacts (components and compositions) can be centrally collected and efficiently managed using the WebComposition/EUD-AL. The requirement is fully fulfilled.

T3: Ease of Use: Ease of use of developed assistance mechanisms has been evaluated during several user studies. The results have been presented in respective chapters of this thesis. The tools apply vocabulary and interactions techniques that address skills and responsibilities of their respective target groups. Composi-

tion platform operates with visual and interactive components that immediately give feedback to its users. ADCE provides a gialog-based UI that guides users through discovery and composition process in the step-by-step fashion. DID applies the PBD technique that does not require any programming skills to be used. Evolution assistance mechanisms use technical vocabulary and enable Information Technology (IT) professionals to gain full control of respective processes. Automated activities can always be manually adjusted to achieve the optimal result. Complexity of utilized concepts and interaction patterns for respective target group is therefore low. The requirement is fully fulfilled.

T4: Fault Tolerance: Configurations produced by Composition Developers can potentially lead to high message traffic on the message bus that in turn can cause data corruption or crash of client-side run-time environment (Web browser). To prevent such situations, the Loop Detection Facilities have been integrated into the run-time environment. They warn users if their configurations can produce looping message behavior and block communication if unsafe traffic is detected. Although the facilities can issue false warnings, they still help to avoid unsafe situations and do it in advance. The requirement is fully fulfilled.

T5: Automation: The WebComposition/EUD assistance mechanisms aim at making development and evolution of DSSs more efficient by automating as much of manual activities as possible. ADCE, DID, WebComposition/EUD-CC, WebComposition/EUD-IE and WebComposition/EUD-AL widely automate finding, usage, extension and maintainance of WebComposition/EUD software artifacts. High-level specification of results to be produced takes place using interactive interfaces. The requirement is therefore fully fulfilled.

Table 8.1 summarizes the results of evaluation and compares the WebComposition/EUD framework with the state-of-the-art approaches presented in Chapter 3.

Table 8.1.: Comparison of the WebComposition/EUD Framework with State-of-the-Art Technologies. Labeling: ++ Requirement fully satisfied, + Requirement partially satisfied, − Requirement not satisfied, / not applicable

Requirement \ Technology	End-User Development	Component-based Development	Model-Driven Development	Web Content Management Systems	Dashboards and Ad-Hoc Reporting Tools	Composition Tools	WebComposition/EUD
End-User Involvement	++	−	−	/	/	/	+
Reuse-orientation	+	++	+	/	/	/	++
Process Conciseness	++	+	+	/	/	/	++
Development Assistance	/	/	/	+	+	+	+
Evolution Assistance	/	/	/	+	+	+	++
Ease of Use	/	/	/	++	++	+	++
Fault Tolerance	/	/	/	++	++	++	++
Automation	/	/	/	+	+	+	++

The WebComposition/EUD framework is characterized by high reuse-orientation and process conciseness. The former is achieved by focusing on development based on discovery of reusable templates and on customization using principles of CDB. The process conciseness is comparable with the one of EUD approaches – the activities of DSS construction do not foresee explicit requirements engineering, design and testing phases. Instead, users build and customize solutions while implicitly testing them and adjusting to their business goals. The involvement of end users into the development and evolution process is higher than in CDB and MDD, but lower than in EUD. End users actively participate in the development process by composing building blocks provided by professional developers. However, integration of new functionality and data-sources as well as their maintenance and management have been assigned to skilled staff.

The WebComposition/EUD toolkit provides the same degree of development and evolution assistance as existing End-User Composition Tools. It follows similar principles and techniques as existing UI mashups and, thus, has a similar functionality set. However, it doesn't fully satisfies the requirement as it limits expressiveness of integration logic and UI configuration. This limitation has been introduced intentionally in accordance to recent research findings that claim generality of composition tools limits their usability (Imran et al., 2012). The toolkit is more powerful than WCMS, Dashboards and Reporting Tools due to the flexibility of Composition Model and extensibility of Artifacts Library. The Ease of Use requirement is, however, fulfilled in the same degree – end users are acting on domain-specific visual components and are offered intuitive dialog-based UIs. In contrast to existing End-User Composition tools, the toolkit doesn't require users to configure data or control flow in their solutions – it is established automatically based on the communication capabilities of WebComposition/EUD components. Fault Tolerance is as well addressed as by existing technologies. Finally, the Automation requirement is better fulfilled due to the particular focus of WebComposition/EUD assistance mechanisms on simplification and automation of development and evolution activities that are crucial in time-pressuring situations.

8.2 Application Scenarios

Applicability of the WebComposition/EUD framework has been tested in several scenarios presented below. The first one, Public Information Screen, describes usage of the Composition Platform to provide aggregated and interactive information on some area of interest. The second one, Collaborative Decision Making, presents composition of DSSs on Microsoft PixelSence[1], a touchscreen device designed as a couch table. Finally, various tools of the framework have been applied to implement demonstrators of the EU FP7 OMELETTE project – the Ermegency Response and First Line Support applications.

8.2.1 Public Information Screen

The WebComposition/EUD toolkit has been applied to build a public information screen of the Distributed and Self-Organizing Systems Group[2] at Faculty of Computer Science, Technische Universität Chemnitz, Germany. The application enabled faculty students, staff and visitors easily navigate through the activities and educational offers of the group. Implementation of the application using WebComposition/EUD framework aimed at raising interest of users in the underlying Web technologies and the group itself.

Two Web applications, consisting of several WebComposition/EUD components provided an aggregated view on group's profile (cf. Figure 8.1). The first composition displayed news and social media posts of group members. Users could scroll through the timelines and inform themselves about recent activities and announcements. The two involved components were autonomous and didn't synchronize their states. The second composition consisted of three components: list of group members, contact details of a group member and list of advised student projects. The components synchronized themselves by means of ICC:

[1] `http://www.microsoft.com/en-us/pixelsense/default.aspx`, Retrieved 19.06.2015

[2] `https://vsr.informatik.tu-chemnitz.de`, Retrieved: 18.06.2015

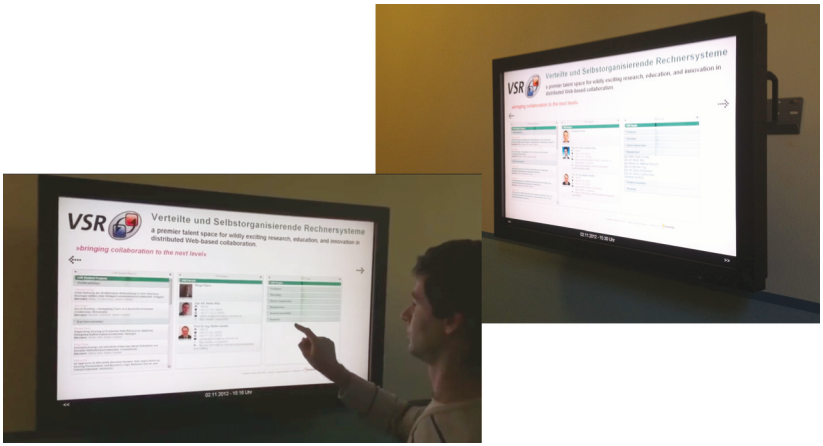


Figure 8.1.: Application of the WebComposition/EUD Framework to build a Public Information Screen

once a group member has been selected in the first component, his/her contact details and advised projects have been display in the others. Selection of a project resulted in the members list to be filtered to display only staff involved into the advising process. Interaction with the application took place over the touch interface of a screen, put in one of the faculty corridors.

8.2.2 Collaborative Decision Making

Design of Microsoft PixelSense as a couch-table with 30 inch large display and up to 52 simultaneous points of contacts make it specifically interesting for education, research, financial services, health, entertainment or travel industry. The WebComposition/EUD toolkit has been partially ported to the PixelSense platform to enable collaborative composition of DSSs using multi-touch capabilities of the device. The conceptual architecture of the composition platform (cf. Section 5.3)

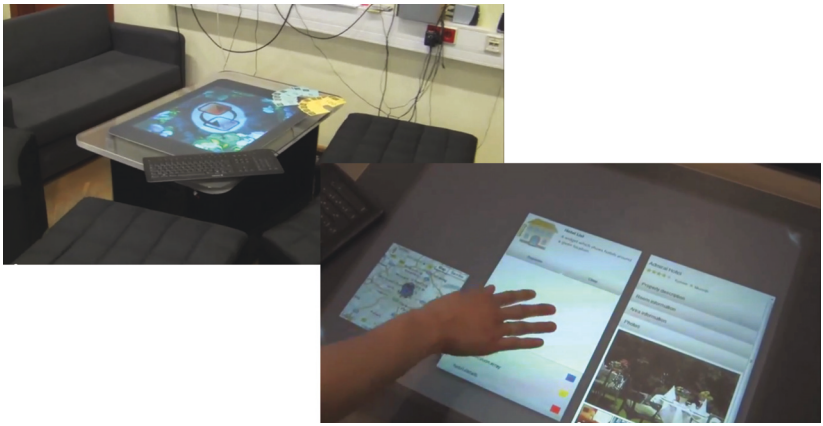


Figure 8.2.: Application of the WebComposition/EUD Framework to Travel Planning Domain

has been implemented using the Microsoft .NET framework[3], Microsoft Surface 1.0 SDK[4], Awesomium.NET browser component[5] and the Plane graphic transformation component[6].

Three dedicated components were implemented to showcast the use of the platform and Microsoft PixelSense in the travel domain (cf. Figure 8.2). An application constructed on-the-fly out of these components should help customers of a travel agency to plan their trip and meet decisions on accommodation, travel route etc. The first component displayed an interactive map and enabled selection of arbitrary locations. The second one provided search results for accommodations around a given location. Finally, the third one showed details of a given location such as description, prices, pictures etc. The three components synchronized their states by means of ICC. Composition Developers could place components from Artifacts Library on the shared canvas and move,

[3] `https://msdn.microsoft.com/en-us/vstudio/aa496123`, Retrieved: 19.06.2015
[4] `http://www.microsoft.com/en-us/download/details.aspx?id=15532`, Retrieved: 19.06.2015
[5] `http://awesomium.com/`, Retrieved: 19.06.2015
[6] `https://github.com/endquote/Plane`, Retrieved: 19.06.2015

resize or configure them. The multi-touch capabilities of the device facilitated the process of collaborative design, configuration and use. Users could duplicate any of them and e.g., compare search results in different locations or chosen accommodation possibilities. The interaction took place using different wiping gestures with one, two or three fingers. The implementation made use of so called identity tags, which are 128-bit codes printed as square patterns. A WebComposition/EUD composition can be assigned to a specific tag and was loaded immediately as soon as an object with the tag was put on the surface of the device.

8.2.3 Telecommunication Dashboards

The composition model of the WebComposition/EUD framework enables efficient integration of event-driven functionalities such as telecommunication services. Being wrapped into WebComposition/EUD components these can be easily combined with data access and processing functions used to support decision making. The framework has been applied in the context of EU FP7 OMELETTE project to build several demonstrators of telecommunication dashboards (cf. Figure 8.3).

The first one, so-called Emergency Cockpit, supported coordination of rescue activities in case of nature catastrophes. It acted as implementation of the motivation scenario in Section 2.1.1 and consisted of several components providing access to map, live water level statistics, feeds from social media, web cameras, contact book, voice and sms calls. The map visualized locations of web cameras, water measurement stations, police departments and origins of social media posts. Using ICC it passed the data on user interactions to other components that synchronized their view correspondingly. The application has been initially populated using ADCE (cf. Section 6.2) and then customized using component browser. Platform awareness and control facilities enabled comparison of water levels in different locations (cf. Section 5.4.4). The second dashboard, so-called First Line Support application, addressed activities related to processing of customer requests received over hotline of a software company. The application consisted of various components to access customer details, order history, order details, sms services, license



Figure 8.3.: Application of the WebComposition/EUD Framework to Implement Telecommunication Dashboards (Source: (OMELETTE Consortium, 2013b))

check and generation. On incoming customer call, the operator could easily find details of the caller, his order history, confirm his identity using sms service and generate a new license if required. The components synchronized their views using ICC. WebComposition/EUD-IE has been used to enrich customer and order access components with ICCI capabilities. Basic components of the application have been interactively discovered by ADCE.

8.3 Summary

Previous sections evaluated WebComposition/EUD assistance mechanisms in isolation. This section analyzed, to which extend the WebComposition/EUD framework as a whole fulfills requirements on an end-user-friendly DSS development method stated in Chapter 2. The analysis showed that all requirements except for End-User Involvement were fully fulfilled. The latter requirement is only partially satisfied due to the intentional decision of separation of responsibilities based on skills of the target groups. The intent behind the decision was to achieve better quality and maintainability of user-produced solutions, even on costs of lower End-User Involvement. The section also presented three domains, were the WebComposition/EUD framework has been practically applied and proved its effectiveness. It has been used to build interactive information screens, collaborative decision making systems and telecommunication dashboards.

9 Conclusions and Outlook

This section summarizes results and contributions of the thesis. It discusses not addressed problems and points out open research challenges.

9.1 Summary of the Thesis

Motivated by the need of timely and ubiquitous access to information in today's competitive economy, the goal of the thesis was set to minimize time, costs and skills required to develop a DSS under time-pressure. The resulting solutions should enable aggregated and interactive access to heterogeneous data sources and functionalities. Development of DSSs directly by decision makers (end users) should lead to fast and cost-efficient solution of their problems.

Experience and knowledge that users collect while using their PCs, tablets or smartphones, is meanwhile deep enough to enable them create their own software instead of being "only" consumers of professionally developed one. Ubiquity of the Internet and growth of the smartphone software industry foster application EUD techniques that build up upon well-understood and widely-applied concepts and interaction patterns. The WebComposition/EUD components can be considered as apps for

the Web and WebComposition/EUD compositions as aggregations of these apps on one screen. These background considerations led to the development approach devised by this thesis.

To address the challenges of time and cost-efficient software development by non-professionals three subordinate research objectives have been identified: 1) definition of principles and methods of DSS construction by end users 2) development of assistance tools to accelerate the construction process and 3) development of mechanisms for cost-efficient maintenance and evolution of user-produced solutions. Based on three motivation scenarios, requirements on appropriate methodology and assisting toolkit have been identified. An analysis of the state-of-the-art technologies resulted in the finding, that the peculiarities of time-constrained end-user development haven't been sufficiently addressed so far.

The proposed solution called WebComposition/EUD consists of five quality-ensuring design measures to establish a systematic development method under time-pressure. The principles promote among others reusability, gentle learning curve and separation of responsibility. The formalisms define WebComposition/EUD conceptual models, related algorithms and architectures. The methods propose techniques to speed-up development and maintenance activities. The process model distinguishes between activities performed by decision makers and by professional programmers. It defines their temporal order and describes handling of produced artifacts. The WebComposition/EUD toolkit consists of a composition platform, development and evolution assistance mechanisms. The composition platform is the core of the framework and provides a visual interactive environment for DSS development. Three platform extensions (ADCE, LDF and DID) aim at accelerating and simplifying the process. Time- and cost-efficient evolution of the solutions is supported by three other mechanisms: WebComposition/EUD-CC, WebComposition/EUD-IE and WebComposition/EUD-AL.

Effectiveness and applicability of the WebComposition/EUD toolkit has been evaluated in several user studies. Participants positively assessed ideas behind the tools and found them useful. The main identified

problems were usability and non-mature UIs of prototype implementations. Analysis of the framework in the context of stated requirements confirmed that the original objectives have been achieved.

9.2 Lessons Learned

Practical application of the developed framework and the presented user studies yielded findings that are not specific to the developed framework but rather interesting for EUD in general:

- Non-programmers easily understand the concept of configurable building blocks that can be combined to more complex solutions.
- Fully-fledged applications can also act as elementary building blocks if wrapped and presented to users appropriately.
- Explanation of capabilities and composition possibilities of provided building blocks is crucial for self-confidence of end users. Symbolic representations and textual descriptions are insufficient for this purpose. Examples, screenshots and explanation videos are better alternatives.
- Users expect to see interfaces and apply interaction techniques that they already know from other related domains (e.g., from the Web or smartphone apps).
- Dialog-based interfaces are effective additions to visual composition environments. They help reduce information overload and increase user's self-confidence while learning new tools.
- Visual programming environments with recommendation and assistance mechanisms enable skilled end users to perform simple schema mapping. Complex data type transformations, however, require programming skills and should be defined by professional developers.

- While automation mechanisms can save time and avoid otherwise error-prone manual activities, transparency is important for their acceptance. Users demand explanations on how an algorithm produced its results (e.g., why particular widgets have been found or why particular communication channels have been established).

9.3 Summary of Contributions

The thesis produced models, algorithms and architectures that enable end-user development of Decision Support Systems under time-constraints. The summary of thesis research contributions is given below:

- Modelling of data-driven DSS as composition of reusable end-user-friendly building blocks called WebComposition/EUD components.
- Definition of a role and a process model for systematic development and evolution of user-created DSSs
- Definition of a conceptual architecture for construction, operation and management of component-based DSSs
- Specification of visualization and interaction mechanisms for end-user-friendly DSS construction process
- Definition of a technology-independent language to describe communication capabilities of components and their behavior in composition
- Development of models, algorithms and description languages to discover and produce partial DSS solutions based on a model of user's business goal

- Development of algorithms to ensure reliability of user-produced solutions and run-time environments
- Specification of models, algorithms and an architecture to automate repeated user input in semantically related contexts
- Development of algorithms and tools to automatically transform proprietary source code of Web-based widgets into WebComposition/EUD components
- Specification of models, algorithms and an architecture to automatically extend Inter-Component Communication Interface of WebComposition/EUD components based on demonstration of desired capabilities.
- Definition and implementation of an architecture for artifact storage and management repository with configurable interface and efficient metadata extraction functionality

9.4 Ongoing and Future Work

The WebComposition/EUD framework addressed several challenges related to time-constrained DSS development by non-programmers. The proposed composition platform aimed at lowering the barrier of software development and hiding complexity of underlying technologies. Assistance mechanisms reduced the time required to development solutions. Tools for evolution support enabled efficient management and maintenance of software artifacts. Future work should focus on both improvements to the efficiency and usability of proposed mechanisms but also explore issues not addressed by the framework.

9.4.1 Toolkit Improvements

Efficiency and usability of composition platform and assistance mechanisms can be improved in several ways. As indicated by user studies non-programmers demand intuitive explanations, guided processes and consistent self-explaining UI. User experience on alternative devices such as mobile phones or tablets should be improved e.g., by applying techniques of responsive design. In the following, improvements to each component of WebComposition/EUD toolkit are discussed.

The composition platform produces interactive dashboard-like solutions that aggregate distributed data sources and functionalities on one screen. With rising number of aggregated components, usability and performance of such solutions might decrease. It should be explored how components can be alternatively visualized or thematically grouped on a screen and which interactions among groups lead to better user experience. Another possible improvement is the possibility of collaborative platform usage that can support joint activities among geographically distributed users. Although Apache Rave enables sharing of composition models among registered users, the running instances are not synchronized in real-time and no communication facilities such as chat or voice calls are supported. An ongoing work is exploring application of Operational Transformation (OT) algorithm to realize conflict-free concurrent usage of WebComposition/EUD compositions (Hertel et al., 2015).

The Automatic Discovery and Composition Engine makes use of explicit composition model and component annotations to automatically discover and build prototype compositions. The approach doesn't provide assistance in creating these annotations, so that they have to be provided manually by Artifact Library Managers and Component Developers. The future work should explore if these annotations can be extracted from textual descriptions of respective artifacts automatically. Furthermore, patterns in user solutions can be mined to automatically populate the goal ontology with new definitions and, thus, continuously extend applicability of the tool.

Loop Detection Facilities help to avoid faulty compositions and configurations. As already discussed in Section 6.3, they produce false positives caused by lack of information on internal business logic of WebComposition/EUD components. However, dedicated annotations e.g., on origin of notification messages, could improve efficiency of the tool and decrease probability of false warnings. Alternatively, automatic source code analysis of WebComposition/EUD components and detection of message retransmissions can be considered.

The Double Input Detector tool can be improved towards higher control of implicitly derived automation rules. The tool focuses on synchronization of input elements only and doesn't enable definition of more complex workflows on top of the GUI. Furthermore, editing of established logical connections is not foreseen, so that users have to remove and demonstrate them anew if first attempts were faulty or unsuccessful. Future work can explore solutions for these deficiencies to make the tool more efficient and easy to use.

The WebComposition/EUD Component Converter enables reuse of source code of proprietary widgets to produce WebComposition/EUD components. Currently the tool support only three source formats (iGoogle, Netvibes, Opera) and should be extend with other adapters. Maintenance of iGoogle and Opera Widgets projects stopped by the moment of writing and new sources of components have to be found. One possibility would be to enable conversion of Apple Dashboard Widgets[1] or Chrome Webapps[2] or even native smartphone apps. Alternatively, conversion of complete Web applications into WebComposition/EUD components can be considered. Promising results have been achieved by recent research e.g., in the context of Java Portlets (Bellas et al., 2008).

The WebComposition/EUD ICCI Extender extends ICCI of components based on observation and manipulation of their GUI. As discussed in Section 7.3, observation of generic DOM node changes and support of JavaScript MVC frameworks can contribute to applicability and ef-

[1] `https://www.apple.com/downloads/dashboard/`,Retrieved: 20.06.2015
[2] `https://chrome.google.com/webstore/category/apps`, Retrieved: 20.06.2015

ficiency of the tool. As in the case of DID, it should be explored how defined patterns can be changed once recorded. The peculiarity of the task is , however, the target group: while DID addressed Composition Developers, the WebComposition/EUD ICCI Extender targets skilled IT staff, namely Component Communication Experts.

The WebComposition/EUD Artifact Library centrally collects and manages access to WebComposition/EUD software artifacts. It stores components, composition models, their metadata and accompanying binary documents. Currently management of artifacts can be performed by the RESTful API only, which makes manual access and review cumbersome. Usability of the tool can be improved by providing a GUI with access to internal search and management functionality. Integration of alternative authentication and authorization methods such as HTTP Basic (J. Franks et al., 1999) or OAuth 2.0 (Hardt, 2012) can further improve reusability and flexibility of the library.

9.4.2 Open Questions

Several research questions have not been tackled by the proposed solution and should be explored in future work.

As already noted in the Chapter 8 end users are not involved into evolution and maintenance activities. The WebComposition/EUD framework considers these activities as too complex to be performed by non-programmers and assigns them to professionals. This decision, however, leads to situations, where users stuck with their solutions and cannot continue, because they miss a component, cannot adjust behavior of existing ones or cannot define more flexible composition logic. According to WebComposition/EUD process model, Composition Developers issue extension requests to the support team, which decides on an optimal way to resolve the problem (development of a new component, extension to existing ones or provisioning of specific transformation functions). Future work should consider transferring some (or all) evolution activities to non-programmers. These could be done by providing case-specific assistance mechanisms (e.g., template-based or guided

component development), simplifying technology (e.g., domain specific or restricted natural languages) or improving communication between users (e.g., collaborative composition).

Another open question is applicability of the proposed tools and interfaces for non-desktop devices. The market share of smartphones, tablets and "smart" devices has significantly increased in the last years and is expected to grow[3]. The WebComposition/EUD framework, however, makes use of programming techniques and interaction patterns that target primarily desktop and notebook devices. Peculiarities of mobile and tablet devices require new methods of information visualization and interaction. In the context of this thesis one prototype for simple multi-touch interactions has been successfully tested (cf. Section 8.2). However, applicability of more complex gestures and other input methods such as using speech interaction should be further explored.

Finally, ensuring performance and security requirements of user-created solutions should be explored in the future work. This thesis considered components as "black boxes", meaning that their efficiency and policy-compliance are ensured by Component Developers only. However, these considerations might not be insufficient, if components are put into composition or if they stem from untrusted sources. Current implementation enables client-side data exchange only, which can lead to performance drawbacks in case of frequent traffic or large amount of transferred data (e.g., documents or media streams). Alternative methods of data exchange and enabling architectures should be investigated. Security threats related to undesired data exposure, access and identity theft should be additionally considered on the level of composition. Although WebComposition/EUD composition model foresees possibility of communication restrictions, it is desirable to devise mechanisms that would enable early detection of improper implementation and run-time behavior. Certification of components, single-sign-on mechanisms and centralized uniform data access are some methods that can be applied to ensure quality and policy-compliance of user-produced solutions.

[3] `http://www.gartner.com/newsroom/id/2408515`, Retrieved: 21.06.2015

Bibliography

Abran, Alain and James W. Moore (2004). *Guide to the Software Engineering Body of Knowledge*. Tech. rep. IEEE, p. 204. URL: http://www.computer.org/portal/web/swebok/htmlformat (Retrieved Mar. 28, 2015).

Aghaee, Saeed and Cesare Pautasso (2013). "Live mashup tools: Challenges and opportunities". In: *Proceedings of the 1st International Workshop on Live Programming (LIVE 2013)*. IEEE Press Piscataway, pp. 1–4.

Aghaee, Saeed and Cesare Pautasso (2014). "End-User Development of Mashups with NaturalMash". In: *Journal of Visual Languages & Computing* 25.4, pp. 414–432.

Aghaee, Saeed, Cesare Pautasso, and Antonella De Angeli (2013). "Natural end-user development of Web Mashups". In: *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* ii, pp. 111–118.

Alves, Alexandre, Assaf Arkin, Sid Askary, Charlton Barreto, et al., eds. (2007). *Web Services Business Process Execution Language Version 2.0*. OASIS. URL: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (Retrieved July 4, 2015).

Amabile, Teresa M, Jennifer S Mueller, William B Simpson, Constance N Hadley, et al. (2002). "Time pressure and creativity in organizations: A longitudinal field study". In: *Harvard Business School Working Paper Series*.

Anton, Ana I. (1997). “Goal Identification and Refinement in the Specification of Software-based Information Systems”. PhD thesis. Atlanta, GA, USA: Georgia Institute of Technology.

Austin, Robert D. (2001). “The Effects of Time Pressure on Quality in Software Development: An Agency Model”. In: *Info. Sys. Research* 12.2, pp. 195–207. URL: http://dx.doi.org/10.1287/isre.12.2.195.9699.

Baker, Dennis, Donald Bridges, Regina Hunter, Gregory Johnson, et al. (2001). *Guidebook to Decision-Making Methods*. Tech. rep. Department of Energy, USA, p. 40.

Bakonyi, Peter, Joost Mak, Alfred Olfert, Pierre-antoine Versini, and Bridget Woods-ballard (2008). *Evacuation and traffic management*. Tech. rep. FLOODsite, p. 178. URL: http://www.floodsite.net/html/cd%5C_task17-19/docs/reports/T17/T17%5C_07%5C_02%5C_Evacuation%5C_and%5C_traffic%5C_management%5C_D17%5C_1%5C_V4%5C_4%5C_P01.pdf.

Beckett, David, Tim Berners-Lee, Eric Prud’hommeaux, and Gavin Carothers (2014). *RDF 1.1 Turtle: Terse RDF Triple Language*. W3C Recommendation. W3C. URL: http://www.w3.org/TR/turtle/ (Retrieved July 4, 2015).

Bellas, Fernando, Inaki Paz, Alberto Pan, and Oscar Díaz (2008). “New Approaches to Portletization of Web Applications”. In: *Handbook of Research on Web Information Systems Quality*. IGI Globa, pp. 250–265.

Berners-Lee, Tim (2009). *Linked-Data Design Issues*. URL: http://www.w3.org/DesignIssues/LinkedData.html (Retrieved May 28, 2015).

Blackwell, AlanF. (2006). “Psychological Issues in End-User Programming”. In: *End User Development*. Ed. by Henry Lieberman, Fabio Paternò, and Volker Wulf. Vol. 9. Human-Computer Interaction Series. Springer Netherlands, pp. 9–30.

Boehm, Barry William (1988). “A Spiral Model of Software Development ans Enhancement”. In: *Computer* 21.5, pp. 61–72.

Booth, David, Hugo Haas, Francis McCabe, Eric Newcomer, et al. (2004). *Web Services Architecture*. W3C. URL: http://www.w3.org/TR/ws-arch/ (Retrieved Mar. 28, 2015).

Boundless (2013). *Boundless: Management*. Boundless.

Boussemart, Yves, Birsen Donmez, ML Cummings, and Jonathan Las Fargeas (2009). "Effects of time pressure on the use of an automated decision support system for strike planning". In: *Proceedings of the 15th International Symposium on Aviation Psychology*.

Brambilla, Marco, Sara Comai, Piero Fraternali, and Maristella Matera (2008). "Designing Web Applications with WebML and WebRatio". In: *Web Engineering Modelling and Implementing Web Applications*, pp. 221–261.

Brown, Alan W. (2000). *Large-Scale, Component Based Development*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Brown, C. M. (1988). *Human-computer Interface Design Guidelines*. Norwood, NJ, USA: Ablex Publishing Corp.

Brynjolfsson, Erik, Lorin Hitt, and Heekyung Kim (2011). "Strength in Numbers: How does data-driven decision-making affect firm performance?" In: *ICIS 2011 Proceedings*, p. 18.

Burnett, Margaret M. (2001). "Software engineering for visual programming languages". In: *Handbook of Software Engineering and Knowledge Engineering*. Ed. by S. K. Chang. Vol. 2. World Scientific Publishing Company, pp. 77–93.

Busemeyer, Jerome R. and Adele Diederich (2002). "Survey of decision field theory". In: *Mathematical social sciences* 43.3, pp. 345–370.

Busemeyer, JeromeR. (1993). "Violations of the Speed-Accuracy Tradeoff Relation". In: *Time Pressure and Stress in Human Judgment and Decision Making*. Ed. by Ola Svenson and A.John Maule. Springer US, pp. 181–193.

Cáceres, Marcos (2012). *Packaged Web Apps (Widgets) - Packaging and XML Configuration*. Web Applications Working Group. URL: `http://www.w3.org/TR/widgets/` (Retrieved Mar. 28, 2015).

Cao, Jill, Yann Riche, Susan Wiedenbeck, Margaret Burnett, and Valentina Grigoreanu (2010). "End-user mashup programming: through the design lens". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, p. 1009.

Capretz, Luiz Fernando (2005). "Y: A New Component-Based Software Life Cycle Model". In: *Journal of Computer Science* 1.1, pp. 76–82.

Ceri, S. (2009). "Search Computing". In: *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*. Vol. 1, pp. 1–1.

Chudnovskyy, Olexiy, Sebastian Brandt, and Martin Gaedke (2011). "Integrating Human-services Using WebComposition/UIX". In: *Proceedings of the Workshop on Posters and Demos Track*. PDT '11. Lisbon, Portugal: ACM, 21:1–21:2.

Chudnovskyy, Olexiy, Christian Fischer, Martin Gaedke, and Stefan Pietschmann (2013). "Inter-Widget Communication by Demonstration in User Interface Mashups". In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 502–505.

Chudnovskyy, Olexiy and Martin Gaedke (2010). "Development of Web 2.0 Applications using WebComposition / Data Grid Service". In: *The Second International Conferences on Advanced Service Computing (Service Computation 2010)*. Ed. by Ali Beklen, Jorge Ejarque, and Wolfgang Gentzsch. **Best Paper Award**. Lisbon, Portugal: IARIA, pp. 55–61.

Chudnovskyy, Olexiy and Martin Gaedke (2012). "End-User-Development and Evolution of Web Applications: The WebComposition EUD Approach". In: *Current Trends in Web Engineering*. Ed. by Michael Grossniklaus and Manuel Wimmer. Vol. 7703. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 221–226.

Chudnovskyy, Olexiy, Hendrik Gebhardt, Frank Weinhold, and Martin Gaedke (2011). "Business Process Integration using Telco Mashups". In: *Procedia Computer Science* 5. The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011), pp. 677–680.

Chudnovskyy, Olexiy, Sebastian Müller, and Martin Gaedke (2012). "Extending Web Standards-Based Widgets towards Inter-Widget Communication". In: *Current Trends in Web Engineering*. Ed. by Michael Grossniklaus and Manuel Wimmer. Vol. 7703. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 93–96.

Chudnovskyy, Olexiy, Tobias Nestler, et al. (2012). “End-User-Oriented Telco Mashups: The OMELETTE Approach”. In: *Proceedings of the 21st International Conference Companion on World Wide Web*. WWW ’12 Companion. New York: ACM, pp. 235–238.

Chudnovskyy, Olexiy, Stefan Pietschmann, Matthias Niederhausen, Vadim Chepegin, et al. (2013). “Awareness and Control for Inter-Widget Communication: Challenges and Solutions”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 114–122.

Chudnovskyy, Olexiy, Frank Weinhold, Hendrik Gebhardt, and Martin Gaedke (2011). “Integration of Telco Services into Enterprise Mashup Applications.” In: *ICWE Workshops*. Ed. by Andreas Harth and Nora Koch. Vol. 7059. Lecture Notes in Computer Science. Springer, pp. 37–48.

Chudnovskyy, Olexiy, Stefan Wild, Hendrik Gebhardt, and Martin Gaedke (2012). “Data Portability Using WebComposition/Data Grid Service”. In: *International Journal on Advances in Internet Technology* 4.3 & 4, pp. 123–132.

Councill, Bill and George T. Heineman (2001). “Component-based Software Engineering”. In: ed. by George T. Heineman and William T. Councill. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. Chap. Definition of a Software Component and Its Elements, pp. 5–19.

Crnković, Ivica (2003). “Component-Based Software Engineering - New Challenges in Software Development”. In: *Journal of Computing and Information Technology* 11, pp. 151–161.

Crnkovic, Ivica, Michel Chaudron, and Stig Larsson (2006). “Component-Based Development Process and Component Lifecycle”. In: *Proceedings of the International Conference on Software Engineering Advances*. ICSEA ’06. Washington, DC, USA: IEEE Computer Society, pp. 44–44.

Csikszentmihalyi, Mihaly (2008). *Flow: The psychology of optimal performance*. Harper Perennial Modern Classics.

Daniel, Florian, Fabio Casati, Boualem Benatallah, and Ming-Chien Shan (2009). "Hosted Universal Composition: Models, Languages and Infrastructure in mashArt". In: *Conceptual Modeling - ER 2009*. Ed. by AlbertoH.F. Laender, Silvana Castano, Umeshwar Dayal, Fabio Casati, and JoséPalazzoM. de Oliveira. Vol. 5829. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 428–443.

Daniel, Florian and Maristella Matera (2009). "Turning Web Applications into Mashup Components: Issues, Models, and Solutions". In: *Web Engineering*. Ed. by Martin Gaedke, Michael Grossniklaus, and Oscar Díaz. Vol. 5648. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 45–60.

Daniel, Florian and Maristella Matera (2014a). "Mashups and End-User Development". In: *Mashups*. Data-Centric Systems and Applications. Springer Berlin Heidelberg, pp. 237–268.

Daniel, Florian and Maristella Matera (2014b). *Mashups: Concepts, Models and Architectures*. Springer-Verlag Berlin Heidelberg, p. 319.

Daniel, Florian, Stefano Soi, Stefano Tranquillini, Fabio Casati, et al. (2011). "Distributed Orchestration of User Interfaces". In: *Information Systems* 37.6, pp. 539–556.

Deufemia, Vincenzo, Chris D'Souza, and Athula Ginige (2013). "Visually modelling data intensive web applications to assist end-user development". In: *Proceedings of the 6th International Symposium on Visual Information Communication and Interaction - VINCI '13*, p. 17.

Díaz, Oscar, Cristóbal Arellano, and Maider Azanza (2013). "A Language for End-user Web Augmentation: Caring for Producers and Consumers Alike". In: *ACM Transactions on the Web (TWEB)* 7.2, 9:1–9:51.

Diederich, a (1997). "Dynamic Stochastic Models for Decision Making under Time Constraints". In: *Journal of mathematical psychology* 41.3, pp. 260–74.

Diederich, Adele and Jerome R. Busemeyer (2003). "Simple matrix methods for analyzing diffusion models of choice probability, choice response time, and simple response time". In: *Journal of Mathematical Psychology* 47.3, pp. 304–322.

Dong, Yifei, Xiaoqun Du, Gerard J. Holzmann, and Scott A. Smolka (2003). "Fighting livelock in the GNU i-protocol: a case study in explicit-state model checking". In: *International Journal on Software Tools for Technology Transfer* 4.4, pp. 505–528.

ECMA International (2015). *ECMAScript Language Specification (Standard ECMA-262)*. URL: `http://www.ecma-international.org/publications/standards/Ecma-262.htm` (Retrieved Mar. 28, 2015).

Eick, Stephen G., Todd L. Graves, Alan F. Karr, J. S. Marron, and Audris Mockus (2001). "Does Code Decay? Assessing the Evidence from Change Management Data". In: *IEEE Trans. Softw. Eng.* 27.1, pp. 1–12. URL: `http://dx.doi.org/10.1109/32.895984`.

Escalona, Maria José and Nora Koch (2007). "Metamodeling the Requirements of Web Systems". In: *Web Information Systems and Technologies*. Ed. by Joaquim Filipe, José Cordeiro, and Vitor Pedrosa. Vol. 1. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 267–280.

Eugster, Patrick Th., Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec (2003). "The Many Faces of Publish/Subscribe". In: *ACM Computing Surveys (CSUR)* 35.2, pp. 114–131.

Fielding, Roy Thomas/Chair-Taylor (2000). "Architectural styles and the design of network-based software architectures". PhD thesis. University of California, Irvine.

Fischer, G., E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev (2004). "Meta-design: A Manifesto for End-user Development". In: *Communications of the ACM* 47.9, pp. 33–37.

Fischer, Gerhard (2009). "End-user development and meta-design: Foundations for cultures of participation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5435 LNCS, pp. 3–14.

Fitzgerald, Brian and Gerard Hartnett (2005). “A Study of the Use of Agile Methods within Intel”. In: *Business Agility and Information Technology Diffusion*. Ed. by RichardL. Baskerville, Lars Mathiassen, Jan Pries-Heje, and JaniceI. DeGross. Vol. 180. IFIP International Federation for Information Processing. Springer US, pp. 187–202.

Forrester Research (2015). *Why Firms Struggle To Analyze More Data*. Tech. rep. November, p. 8.

Forsberg, Kevin; and Harold Mooz (1994). “The Relationship of System Engineering to the Project Cycle”. In: *The 12th INTERNET World Congress on Project Management*, p. 12.

Förster, Jens, E. Tory Higgins, and Amy Taylor Bianco (2003). “Speed/accuracy decisions in task performance: Built-in trade-off or separate strategic concerns?” In: *Organizational Behavior and Human Decision Processes* 90.1, pp. 148–164.

Franks, J, P Hallam-Baker, and J Hostetler (1999). “RFC 2617: HTTP Authentication: Basic and Digest Access Authentication”. In: *Internet RFCs*, pp. 1–35. URL: `https://www.ietf.org/rfc/rfc2617.txt` (Retrieved Mar. 28, 2015).

Franks, J., P. Hallam-Baker, J. Hostetler, S. Lawrence, et al. (1999). *HTTP Authentication: Basic and Digest Access Authentication*. Internet Engineering Task Force. URL: `http://www.ietf.org/rfc/rfc2617.txt` (Retrieved Mar. 28, 2015).

Fuggetta, Alfonso (1993). “A Classification of CASE Technology”. In: *Computer* 26.12, pp. 25–38. URL: `http://dx.doi.org/10.1109/2.247645`.

Gachet, Alexandre and Pius Haettenschwiler (2006). “Development Processes of Intelligent Decision-making Support Systems: Review and Perspective”. English. In: *Intelligent Decision-making Support Systems*. Decision Engineering. Springer London, pp. 97–121. URL: `http://dx.doi.org/10.1007/1-84628-231-4_6`.

Gaedke, Martin and Guntram Gräf (2001). “Development and Evolution of Web-Applications Using the WebComposition Process Model”. In: *Web Engineering* 2016, pp. 58–76.

Gaedke, Martin and Jörn Rehse (2000). “Supporting compositional reuse in component-based Web engineering”. In: *Proceedings of the 2000 ACM symposium on Applied*, pp. 927–933.

Gaffney, J. E. and T. A. Durek (1989). “Software Reuse&Mdash;Key to Enhanced Productivity: Some Quantitative Models”. In: *Inf. Softw. Technol.* 31.5, pp. 258–267.

Ghiani, Giuseppe, Fabio Paternò, and L Spano (2011). “Creating mashups by direct manipulation of existing web applications”. In: *End-User Development*, pp. 42–52. URL: http://www.springerlink.com/index/D5X8150851042663.pdf.

Ghosh, S., S. Ramaswamy, and R.P. Jetley (2013). “Towards Requirements Change Decision Support”. In: *Software Engineering Conference (APSEC), 2013 20th Asia-Pacific*. Vol. 1, pp. 148–155.

Gigerenzer, Gerd and Peter M. Todd (2000). *Simple Heuristics That Make Us Smart (Evolution and Cognition)*. Oxford University Press, U.S.A., p. 438.

Gluchowski, Peter, Roland Gabriel, and Carsten Dittmar (2008). *Management Support Systeme und Business Intelligence*. Springer, p. 432.

Gordon, Paul M K, Ken Barker, and Christoph W. Sensen (2010). “Programming-by-example meets the Semantic Web: Using ontologies and web services to close the semantic gap”. In: *Proceedings - 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2010*, pp. 133–140.

Guenther, P. and T. Showalter (2008). *Sieve: An Email Filtering Language*. Internet Engineering Task Force. URL: http://www.ietf.org/rfc/rfc5228.txt (Retrieved Mar. 28, 2015).

Güth, Werner, Rolf Schmittberger, and Bernd Schwarze (1982). “An experimental analysis of ultimatum bargaining”. In: *Journal of Economic Behavior & Organization* 3.4, pp. 367–388.

Haas, Hugo and Allen Brown (2004). *Web Services Glossary*. W3C. URL: http://www.w3.org/TR/ws-gloss/ (Retrieved Mar. 28, 2015).

Hadley, Marc J. (2009). *Web Application Description Language*. URL: http://www.w3.org/Submission/wadl/ (Retrieved Mar. 28, 2015).

Hardt, Dick (2012). *The OAuth 2.0 Authorization Framework*. RFC 6749. Fremont, CA, USA: RFC Editor. URL: http://tools.ietf.org/html/rfc6749 (Retrieved July 4, 2015).

Harris, Robert (2012). *Introduction to Decision Making*. URL: http://www.virtualsalt.com/crebook5.htm (Retrieved Mar. 28, 2015).

Hartmann, Björn, Leslie Wu, Kevin Collins, and Scott R. Klemmer (2007). "Programming by a Sample: Rapidly Creating Web Applications with D.Mix". In: *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*. UIST '07. Newport, Rhode Island, USA: ACM, pp. 241–250.

Hedgebeth, Darius (2007). "Data-driven decision making for the enterprise: an overview of business intelligence applications". In: *VINE* 37.4, pp. 414–420. eprint: http://dx.doi.org/10.1108/03055720710838498. URL: http://dx.doi.org/10.1108/03055720710838498.

Heineman, George T. and William T. Councill, eds. (2001). *Component-based Software Engineering: Putting the Pieces Together*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Henschen, Douglas (2012). *InformationWeek 2012 Enterprise Applications Survey*. Tech. rep. InformationWeek, p. 33. URL: http://reports.informationweek.com/abstract/1/8954/Application-Performance-Optimization/research-2012-enterprise-applications-survey-.html?cid=pub_analyt__iwk_20120813 (Retrieved July 4, 2015).

Hertel, Michael (2012). "Analyse von existierenden Widget-Formaten und Entwicklung eines Verfahrens zu deren Transformation in das W3C-Format". Bachelorarbeit. Technische Universität Chemnitz.

Hertel, Michael, Alexey Tschudnowsky, and Martin Gaedke (2015). "Conflict Resolution in Collaborative User Interface Mashups". In: *Engineering the Web in the Big Data Era*. Ed. by Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe. Vol. 9114. Lecture Notes in Computer Science. Springer International Publishing, pp. 659–662.

Hickson, Ian (2013). *Web Storage*. Ed. by Ian Hickson. W3C. URL: http://www.w3.org/TR/webstorage/ (Retrieved Apr. 7, 2015).

Hickson, Ian, Robin Berjon, Steve Faulkner, Travis Leithead, et al. (2014). *HTML5: A vocabulary and associated APIs for HTML and XHTML*. W3C. URL: http://www.w3.org/TR/2014/CR-html5-20140731/ (Retrieved July 5, 2015).

Hodgkinson, Gerard P and William H Starbuck (2008). *The Oxford handbook of organizational decision making*. Oxford Handbooks Online.

Hurwitz, Judith, Fern Halper, and Marcia Kaufman (2005). *Dashboards – Enabling Insight and Action*. Tech. rep. Hurwitz & Associates, p. 17.

Hwang, Mark I. (1994). "Decision making under time pressure: A model for information systems research". In: *Information & Management* 27.4, pp. 197–203. URL: http://www.sciencedirect.com/science/article/pii/0378720694900485.

Imran, Muhammad, Felix Kling, Stefano Soi, Florian Daniel, et al. (2012). "ResEval Mash: A Mashup Tool for Advanced Research Evaluation". In: *Proceedings of the 21st International Conference Companion on World Wide Web*. WWW '12 Companion. Lyon, France: ACM, pp. 361–364.

Inacio, Chris (1998). *Software Fault Tolerance*. URL: http://users.ece.cmu.edu/~koopman/des%5C_s99/sw%5C_fault%5C_tolerance/ (Retrieved Mar. 28, 2015).

International Organization for Standardization (1998). *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability*. Tech. rep. International Organization for Standardization (ISO), p. 22. URL: http://www.iso.org/iso/home/store/catalogue%5C_tc/catalogue%5C_detail.htm?csnumber=16883 (Retrieved Mar. 28, 2015).

Irvine, C.A. (1976). "The Software Engineer: Role, Responsibilities and Education". English. In: *Software Engineering Education*. Ed. by AnthonyI. Wasserman and Peter Freeman. Springer New York, pp. 23–27. URL: http://dx.doi.org/10.1007/978-1-4612-9898-4_6.

Isaksson, Erik and Matthias Palmer (2010). "Usability and inter-widget communication in PLEs". In: *MUPPLE-10*. Aachen: Sun SITE Central Europe, RWTH Aachen.

Isenberg, Daniel J (1984). “How Senior Managers Think”. In: *Harvard Business Review* 62, pp. 81–90.

The Economics Of Software Maintenance in the Twenty First Century Version (2006). URL: http://www.compaid.com/caiinternet/ezine/%20capersjones-maintenance.pdf (Retrieved Mar. 28, 2015).

Juhnke, Ernst, Tim Dörnemann, Sebastian Kirch, Dominik Seiler, and Bernd Freisleben (2010). “SimpleBPEL: Simplified modeling of BPEL workflows for scientific end users”. In: *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010)*, pp. 137–140.

Kaye, Doug (2003). *Loosely Coupled: The Missing Pieces of Web Services*. RDS Press.

Kielstra, Paul, Denis McCauley, and Mike Kenny (2007). *In Search of Clarity. Unravelling the Complexities of Executive Decision-Making*. Tech. rep. Economist Intelligence Unit, p. 23.

Kim, Seung Han and Jae Wook Jeon (2007). “Programming LEGO Mindstorms NXT with visual programming”. In: *ICCAS 2007 - International Conference on Control, Automation and Systems*, pp. 2468–2472.

Kim, Won, Ok-Ran Jeong, and Sang-Won Lee (2010). “On Social Web Sites”. In: *Inf. Syst.* 35.2, pp. 215–236. URL: http://dx.doi.org/10.1016/j.is.2009.08.003.

Kleppe, Anneke, Jos Warmer, and Wim Bast (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Vol. 83. Addison-Wesley Longman Publishing Co., Inc, p. 192.

Ko, AJ and BA Myers (2004). “Designing the whyline: a debugging interface for asking questions about program behavior”. In: *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*. Vol. 6. 1, pp. 151–158.

Ko, Andrew J., Robin Abraham, Laura Beckwith, Alan Blackwell, et al. (2011). “The State of the Art in End-user Software Engineering”. In: *ACM Comput. Surv.* 43.3, 21:1–21:44. URL: http://doi.acm.org/10.1145/1922649.1922658.

Koch, Nora and Fast GmbH (2006). “Transformation Techniques in the Model-Driven Development Process of UWE”. In: *Workshop Proceedings of the Sixth International Conference on Web Engineering (ICWE'06).ACM*, p. 3.

Koch, Nora, Alexander Knapp, Gefei Zhang, and Hubert Baumeister (2007). “UML-based Web Engineering: An Approach Based on Standards”. In: *Web Engineering: Modelling and Implementing Web Applications*. Ed. by Gustavo Rossi, Oscar Pastor, Daniel Schwabe, and Luis Olsina. Springer Science & Business Media. Chap. 7, p. 462.

Kocher, Martin G. and Matthias Sutter (2006). “Time is money—Time pressure, incentives, and the quality of decision-making”. In: *Journal of Economic Behavior & Organization* 61.3, pp. 375–392. URL: http://www.sciencedirect.com/science/article/pii/S0167268105001873.

Krug, Michael, Fabian Wiedemann, and Martin Gaedke (2013). “Media Enrichment on Distributed Displays by Selective Information Presentation: A First Prototype”. English. In: *Current Trends in Web Engineering*. Ed. by QuanZ. Sheng and Jesper Kjeldskov. Vol. 8295. Lecture Notes in Computer Science. Springer International Publishing, pp. 51–53.

Kruglanski, Arie W and Tallie Freund (1983). “The freezing and unfreezing of lay-inferences: Effects on impressional primacy, ethnic stereotyping, and numerical anchoring”. In: *Journal of Experimental Social Psychology* 19.5, pp. 448–468.

Kumar, Rajeev (2013). “Efficient Customization of Software Applications of an Organization”. In: *International Journal of Business and Social Science* 4.11.

Lau, Kung-Kiu, F.M. Taweel, and C.M. Tran (2011). “The W Model for Component-Based Software Development”. In: *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pp. 47–50.

Lau, Tessa (2001). “Programming by Demonstration : a Machine Learning Approach”. In: *Journal of Experimental* 16, pp. 161–188.

Le Hors, Arnaud, Philippe Le Hégaret, Lauren Wood, Gavin Thomas Nicol, et al. (2004). *Document Object Model (DOM) Level 3 Core Specification*. W3C. URL: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407 (Retrieved July 5, 2015).

Lehman, Meir M. and Juan F. Ramil (2003). “Software Evolution: Background, Theory, Practice”. In: *Inf. Process. Lett.* 88.1-2, pp. 33–44. URL: http://dx.doi.org/10.1016/S0020-0190(03)00382-X.

Leone, Stefania, Alexandre De Spindler, Moira C. Norrie, and Dennis McLeod (2013). “Integrating component-based web engineering into content management systems”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7977 LNCS, pp. 37–51.

Leshed, Gilly, Eben M. Haber, Tara Matthews, and Tessa Lau (2008). “CoScripter: Automating & Sharing How-to Knowledge in the Enterprise”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’08. Florence, Italy: ACM, pp. 1719–1728.

Leue, Stefan, Alin Ştefănescu, and Wei Wei (2006). “A Livelock Freedom Analysis for Infinite State Asynchronous Reactive Systems”. English. In: *CONCUR 2006 – Concurrency Theory*. Ed. by Christel Baier and Holger Hermanns. Vol. 4137. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 79–94.

Lieberman, Henry (2001). *Your Wish is My Command*. Ed. by Henry Lieberman. Morgan Kaufmann.

Lieberman, Henry, Fabio Paternó, Markus Klann, Fabio Paternò, and Volker Wulf (2006). “End-user development: An emerging paradigm”. In: *End user development* 9, pp. 1–8.

Lim, Wayne C. (1994). “Effects of Reuse on Quality, Productivity, and Economics”. In: *IEEE Software* 11.5, pp. 23–30. URL: http://dx.doi.org/10.1109/52.311048.

Litvinova, Evgenia, Markku Laine, and Petri Vuorimaa (2012). “XIDE: Expanding End-User Web Development”. In: *The Eighth International Conference on Web Information Systems and Technologies (WEBIST’12)*, pp. 123–128.

Lizcano, David, Javier Soriano, Marcos Reyes, and Juan J. Hierro (2008). “EzWeb/FAST: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming "Ubiquitous SOA"”. In: *Proceedings - The 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM 2008*, pp. 488–495.

Loveman, Gary (2003). “Diamonds in the Data Mine”. In: *Harvard Business Review* 81.

Lucchi, Roberto, Michel Millot, and Christian Elfers (2008). “Resource Oriented Architecture and REST”. In: *Assessment of impact and advantages on INSPIRE Ispra European Communities*, p. 16.

Maccomascaigh, Mick, Mark R. Gilbert, Jim Murphy, and Gavin Tay (2013). *Magic Quadrant for Web Content Management*. Tech. rep. Gartner, Inc. URL: `https://www.gartner.com/doc/2565615/magic-quadrant-web-content-management` (Retrieved Mar. 28, 2015).

Mackay, Wendy E. (1990). “Users And Customizable Software: A Co-Adaptive Phenomenon”. Ph.D. Thesis. Sloan School of Management, Massachusetts Institute of Technology.

Madhavji, Nazim H, Juan Fernandez-Ramil, and Dewayne Perry (2006). *Software evolution and feedback: Theory and practice*. John Wiley & Sons.

Martin, Robert C (2002). *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall Computer, p. 529.

Matera, Maristella, Matteo Picozzi, Michele Pini, and Marco Tonazzo (2013). “PEUDOM: A Mashup Platform for the End User Development of Common Information Spaces”. English. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 494–497.

Maule, A John and Isabel Andrade (1997). “The effects of time pressure on decision making: how harassed managers cope”. In: *Decision Making and Problem Solving (Digest No: 1997/366), IEE Colloquium on*. IET, pp. 4–1.

Mehandjiev, Nikolay and Leonardo Bottaci (1996). “User-enhanceability for organisational information systems through visual programming”. English. In: *Advanced Information Systems Engineering*. Ed. by Panos Constantopoulos, John Mylopoulos, and Yannis Vassiliou. Vol. 1080. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 432–456.

Mehandjiev, Nikolay, Freddy Lécué, Usman Wajid, and Abdallah Namoun (2010). “Assisted service composition for end users”. In: *Proceedings - 8th IEEE European Conference on Web Services, ECOWS 2010*, pp. 131–138.

Mehandjiev, Nikolay, Alistair Sutcliffe, and Darren Lee (2006). "Organizational View of End-User Development". In: *End User Development*. Ed. by Henry Lieberman, Fabio Paternò, and Volker Wulf. Vol. 9. Human-Computer Interaction Series. Springer Netherlands, pp. 371–399.

Meliá, Santiago, Andreas Kraus, and Nora Koch (2005). "MDA Transformations Applied to Web Application Development". In: *Web Engineering*. Ed. by David Lowe and Martin Gaedke. Vol. 3579. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 465–471.

Miah, Shah Jahan and Footscray Park (2012). "An Emerging Decision Support Systems Technology for Disastrous Actions Management". In: *Emerging Informatics - Innovative Concepts and Applications*. Ed. by Shah Jahan Miah. InTech, pp. 101–110.

Microsoft (2015a). *COM Component Object Model Technologies*. URL: `http://www.microsoft.com/com` (Retrieved Mar. 30, 2015).

Microsoft (2015b). *Microsoft .NET*. URL: `http://www.microsoft.com/net` (Retrieved Mar. 15, 2015).

Miller, Michael (2008). *Cloud computing: Web-based applications that change the way you work and collaborate online*. Que publishing, p. 293.

Mørch, Anders I, Gunnar Stevens, Markus Won, Markus Klann, et al. (2004). "Component-Based Technologies for End-User Development". In: *Communications of the ACM* 44.9, pp. 59–62.

Mosterd, Igor, Christel G Rutte, and Igor Mosterd (2000). "Effects of time pressure and accountability to constituents on negotiation". In: *International Journal of Conflict Management* 11(2000).3, pp. 227–247.

Myers, Brad, John Pane, and Andy Ko (2004). "Natural Programming Languages and Environments". In: *Communications of the ACM* 47.9, pp. 47–52.

Nah, Fiona Fui-Hoon (2004). "A study on tolerable waiting time: how long are Web users willing to wait?" In: *Behaviour & Information Technology* 23.3, pp. 153–163.

Nardi, Bonnie A (1993). *A Small Matter of Programming: Perspectives on End User Computing*. Vol. 26. MIT Press, p. 162.

Nestler, Tobias, Abdallah Namoun, and Alexander Schill (2011). "End-user development of service-based interactive web applications at the presentation layer". In: *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 197–206.

Ngu, A.H.H., M.P. Carlson, Q.Z. Sheng, and Hye-young Paik (2010). "Semantic-Based Mashup of Composite Applications". In: *IEEE Transactions on Services Computing* 3.1, pp. 2–15.

Nightingale, Jim (2007). *Think Smart - Act Smart: Avoiding The Business Mistakes That Even Intelligent People Make*. 1st ed. Wiley, p. 176.

Nottingham, M and R Sayre (2005). *The Atom Syndication Format*. URL: `http://www.ietf.org/rfc/rfc4287.txt` (Retrieved Mar. 28, 2015).

OMELETTE Consortium (2013a). *D2.3 - Final Specification of Mashup Description Language and Telco Mashup Architecture*. Public Deliverable. The OMELETTE Project (FP7/2010-2013 grant agreement Nr. 257635).

OMELETTE Consortium (2013b). *D7.3 - Final Demonstrators*. Public Deliverable. The OMELETTE Project (FP7/2010-2013 grant agreement Nr. 257635).

OMELETTE Consortium (2013c). *D7.4 - Evaluations of Demonstrators Report*. Public Deliverable. The OMELETTE Project (FP7/2010-2013 grant agreement Nr. 257635).

OMG (2015). *CORBA*. URL: `http://www.corba.org/` (Retrieved Mar. 30, 2015).

OpenAjax Alliance (2009). *OpenAjax Hub 2.0 Specification*. URL: `http://www.openajax.org/member/wiki/OpenAjax%5C_Hub%5C_2.0%5C_Specification` (Retrieved Apr. 13, 2015).

OpenSocial and Gadgets Specification Group (2014). *OpenSocial Specification 2.5.0*. URL: `http://opensocial.github.io/spec/2.5/OpenSocial-Specification.xml` (Retrieved July 4, 2015).

O'Reilly, Tim (2005). *What is web 2.0? Design Patterns and Business Models for the Next Generation of Software*. URL: `http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1` (Retrieved July 4, 2015).

Orts, Daryl (2005). “Dashboard Development and Deployment”. In: *KM World* 14.1, S17. URL: http://proquest.umi.com/pqdweb?did=783713271%5C&Fmt=7%5C&clientId=65345%5C&RQT=309%5C&VName=PQD (Retrieved July 4, 2015).

OSGi Alliance (2015). *OSGi Alliance Specifications*. URL: http://www.osgi.org/Specifications/HomePage (Retrieved Mar. 30, 2015).

Panko, Raymond R (2008). “What We Know About Spreadsheet Errors”. In: *Journal of End User Computing’s* 10, pp. 15–21.

Pedrinaci, Carlos, Dong Liu, Maria Maleshkova, David Lambert, et al. (2010). “IServe: A linked services publishing platform”. In: *Proceedings of 1st International Workshop on Ontology Repositories and Editors for the Semantic Web (ORES 2010)*. Vol. 596, pp. 71–82.

Pérez, Francisca, Pedro Valderas, and Joan Fons (2011). “Allowing end-users to participate within model-driven development approaches”. In: *Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011*, pp. 187–190.

Pietschmann, Stefan, Carsten Radeck, Klaus Meißner, and Klaus Meissner De (2011). “Semantics-Based Discovery, Selection and Mediation for Presentation-Oriented Mashups”. In: *Proceedings of the 5th International Workshop on Web APIs and Service Mashups*. ACM New York, 7:1–7:8.

Pietschmann, Stefan, Martin Voigt, and Klaus Meißner (2012). “Rich Communication Patterns for Mashups”. English. In: *Web Engineering*. Ed. by Marco Brambilla, Takehiro Tokuda, and Robert Tolksdorf. Vol. 7387. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 315–322.

Power, D (2004). “Specifying An Expanded Framework for Classifying and Describing Decision Support Systems”. In: *Communications of the Association for Information Systems* 13, pp. 158–166. URL: http://aisel.aisnet.org/cais/vol13/iss1/13 (Retrieved July 4, 2015).

Power, Daniel J. (2002). *Decision Support Systems: Concepts and Resources for Managers*. Quorum Books, p. 272.

Power, Daniel J. (2008). “Decision Support Systems Concept”. In: *Encyclopedia of Decision Making and Decision Support Technologies*. Ed. by Patrick Humphreys Frederic Adam. Information Science Reference, pp. 232–235.

Radeck, Carsten, Gregor Blichmann, and Klaus Meißner (2013). “CapView – Functionality-Aware Visual Mashup Development for Non-programmers”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 140–155.

Reich, Siegfried, Gerti Kappel, and Birgit Pr (2006). *Web Engineering: The Discipline of Systematic Development of Web Applications*. Ed. by Gerti Kappel, Birgit Proll, Siegfried Reich, and Werner Retschitzegger. Vol. 4. John Wiley & Sons, Ltd, p. 387.

Repenning, Alexander and Andri Ioannidou (2006). “What makes end-user development tick? 13 design guidelines”. In: *End User Development*, pp. 1–41.

Rivero, José Matías, Sebastian Heil, Julián Grigera, Martin Gaedke, and Gustavo Rossi (2013). “MockAPI: An Agile Approach Supporting API-first Web Application Development”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 7–21.

Robie, Jonathan, Don Chamberlin, Michael Dyck, and John Snelson (2011). *XQuery 3.0: An XML Query Language*. W3C. URL: `http://www.w3.org/TR/xquery-30` (Retrieved Mar. 28, 2015).

Rode, Jochen, Yogita Bhardwaj, ManuelA. Pérez-Quiñones, MaryBeth Rosson, and Jonathan Howarth (2005). “As Easy as "Click": End-User Web Engineering”. In: *Web Engineering*. Ed. by David Lowe and Martin Gaedke. Vol. 3579. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 478–488.

Rosson, M.B., H. Sinha, M. Bhattacharya, and Dejin Zhao (2007). “Design Planning in End-User Web Development”. In: *IEEE Symposium on Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007*. IEEE, pp. 189–196.

Roy Chowdhury, Soudip, Olexiy Chudnovskyy, Matthias Niederhausen, Stefan Pietschmann, et al. (2013). "Complementary Assistance Mechanisms for End User Mashup Composition". In: *Proceedings of the 22nd International Conference on World Wide Web Companion*. WWW '13 Companion. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, pp. 269–272.

Roy Chowdhury, Soudip, Carlos Rodríguez, Florian Daniel, and Fabio Casati (2012). "Baya: Assisted Mashup Development As a Service". In: *Proceedings of the 21st International Conference Companion on World Wide Web*. WWW '12 Companion. Lyon, France: ACM, pp. 409–412.

Royce, W. W. (1987). "Managing the Development of Large Software Systems: Concepts and Techniques". In: *Proceedings of the 9th International Conference on Software Engineering*. ICSE '87. Monterey, California, USA: IEEE Computer Society Press, pp. 328–338.

Rümpel, Andreas, Carsten Radeck, Gregor Blichmann, Alexander Lorz, and Klaus Meißner (2011). "Towards Do-It-Yourself Development of Composite Web Applications". In: *International Conference on Internet Technologies & Society ITS 2011*, pp. 330–332.

Saleem, Rizwan, Anwar ul Haq Shah, and Muhammad Waqas (2011). "Effect of time pressure and human judgment on decision making in three public sector organizations of Pakistan". In: *International Journal of Human Sciences* 8.1, p. 1188.

Sambra, Andrei, Henry Story, and Tim Berners-Lee (2014). *WebID 1.0: Web Identity and Discovery*. W3C Editor's Draft. W3C. URL: `http://www.w3.org/TR/turtle/` (Retrieved July 4, 2015).

Sauter, Vicki L (2014). *Decision support systems for business intelligence*. John Wiley & Sons.

Scacchi, Walt (2006). "Understanding open source software evolution". In: *Software Evolution and Feedback: Theory and Practice*, pp. 181–206.

Schadler, Ted, Stephen Powers, and Steven Kesler (2015). *The Forrester Wave: Web Content Management Systems, Q1 2015*. Tech. rep. Forrester.

Schmidt, Douglas C (1999). “Why software reuse has failed and how to make it work for you”. In: *C++ Report* 11.1, p. 1999.

Schmiedel, Philipp (2013). “Eine gemeinsame Sprache finden: Unterstützung der Datentransformation bei Inter-Widget-Kommunikation”. Diplomarbeit. Technische Universität Chemnitz.

Schwabe, Daniel, Guilherme Szundy, Sabrina Silva De Moura, and Fernanda Lima (2004). “Design and Implementation of Semantic Web Applications”. In: *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*.

Segal, Judith and Chris Morris (2008). “Developing scientific software”. In: *IEEE software* 25.4, pp. 18–20.

Shawish, Ahmed and Maria Salama (2014). “Cloud Computing: Paradigms and Technologies”. In: *Inter-cooperative Collective Intelligence: Techniques and Applications*. Springer, pp. 39–67.

Skrobo, Daniel (2009). “Widget-Oriented Consumer Programming”. In: *AUTOMATIKA: Journal for Control, Measurement, Electronics, Computing and Communications* 50.3-4, pp. 252–264.

Smith, D Leasel, Dean G Pruitt, and Peter J Carnevale (1982). “Matching and mismatching: The effect of own limit, other’s toughness, and time pressure on concession rate in negotiation.” In: *Journal of Personality and Social Psychology* 42.5, p. 876.

Solana, E., V. Baggiolini, M. Ramluckun, and J. Harms (1996). “Automatic and Reliable Elimination of E-mail Loops Based on Statistical Analysis”. In: *Proceedings of the 10th USENIX Conference on System Administration*. LISA ’96. Chicago, IL: USENIX Association, pp. 139–144.

Souza Bomfim, MauricioHenrique de and Daniel Schwabe (2011). “Design and Implementation of Linked Data Applications Using SHDM and Synth”. In: *Web Engineering*. Ed. by Sören Auer, Oscar Díaz, and GeorgeA. Papadopoulos. Vol. 6757. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 121–136.

Spahn, Michael and Volker Wulf (2009). “End-User Development of Enterprise Widgets”. In: *Second International Symposium on End User Development (IS-EUD 2009)*. Springer, pp. 106–125.

Spira, Jonathan B.. and Joshua B. Feintuch (2005). *The Cost of Not Paying Attention: How Interruptions Impact Knowledge Worker Productivity*. Tech. rep. Basex. URL: http://www.brigidschulte.com/wp-content/uploads/2014/02/costofnotpayingattention.basexreport-2.pdf (Retrieved July 4, 2015).

Stiemerling, O, H Kahler, and V Wulf (1997). “How to Make Software Softer - Designing Tailorable Applications”. In: *Symposium on Designing Interactive Systems*, pp. 365–376.

Stuhlmacher, Alice F and Matthew V Champagne (2000). “The impact of time pressure and information on negotiation process and decisions”. In: *Group Decision and Negotiation* 9, pp. 471–491.

Sugiura, Atsushi, Atsushi Sugiura, Yoshiyuki Koseki, and Yoshiyuki Koseki (1996). “Simplifying Macro Definition in Programming by Demonstration”. In: *9th Annual ACM Symposium on User-Interface Software and Technology*, pp. 173–182.

Sumathi, S. and P. Surekha (2007). *LabVIEW based advanced instrumentation systems*. Springer Berlin Heidelberg, p. 728.

Sun Microsystems (2003). *JSR-168 Portlet Specification 1.0*. URL: https://jcp.org/aboutJava/communityprocess/final/jsr168/ (Retrieved June 9, 2015).

Sun Microsystems (2009). *JSR-286 Portlet Specification 2.0*. URL: http://download.oracle.com/otndocs/jcp/portlet-2.0-fr-oth-JSpec/ (Retrieved Apr. 13, 2015).

Sun Microsystems, Inc (2009). *JSR-000318 Enterprise JavaBeans(tm) ("Specification")*. URL: http://download.oracle.com/otndocs/jcp/ejb-3.1-fr-eval-oth-JSpec/ (Retrieved Mar. 30, 2015).

Sutter, Matthias, Martin Kocher, and Sabine Strauß (2003). “Bargaining under time pressure in an experimental ultimatum game”. In: *Economics Letters* 81, pp. 341–347.

Tai, Kuo-Chung (1994). “Definitions and Detection of Deadlock, Livelock, and Starvation in Concurrent Programs”. In: *Proceedings of the 1994 International Conference on Parallel Processing - Volume 02*. ICPP ’94. Washington, DC, USA: IEEE Computer Society, pp. 69–72.

Tam, R. Chung-Man, David Maulsby, and Angel R. Puerta (1998). “U-TEL: A Tool for Eliciting User Task Models from Domain Experts”. In: *Proceedings of the 3rd International Conference on Intelligent User Interfaces*. IUI ’98. San Francisco, California, USA: ACM, pp. 77–80.

Tanimoto, Steven L. (1990). “VIVA: A visual language for image processing”. In: *Journal of Visual Languages & Computing* 1, pp. 127–139.

Tarjan, Robert E (1983). *Data Structures and Network Algorithms*. Philadelphia: SIAM, p. 131.

Thomas H. Davenport, Jeanne G. Harris (2007). *Competing on Analytics: The New Science of Winning*. Harvard Business Press, p. 218.

Tietz, Vincent, Stefan Pietschmann, Gregor Blichmann, Klaus Meißner, et al. (2011). “Towards Task-based Development of Enterprise Mashups”. In: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*. iiWAS ’11. Ho Chi Minh City, Vietnam: ACM, pp. 325–328.

Tockey, S. (1999). “Recommended skills and knowledge for software engineers”. In: *Software Engineering Education and Training, 1999. Proceedings. 12th Conference on*, pp. 168–176.

Trinkfass, Gabriele (1997). “Testing the empirical relevance of the innovation spiral”. English. In: *The Innovation Spiral*. Deutscher Universitätsverlag, pp. 108–259. URL: http://dx.doi.org/10.1007/978-3-663-09041-0_4.

Tschudnowsky, Alexey and Martin Gaedke (2015). “Loop Discovery in Publish-Subscribe-Based User Interface Mashups”. In: *Engineering the Web in the Big Data Era*. Ed. by Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe. Vol. 9114. Lecture Notes in Computer Science. Springer International Publishing, pp. 683–686.

Tschudnowsky, Alexey, Michael Hertel, Fabian Wiedemann, and Martin Gaedke (2014). “Towards Real-time Collaboration in User Interface Mashups”. In: *ICE-B 2014 - Proceedings of the 11th International Conference on e-Business*. Vienna, Austria, pp. 193–200.

Tschudnowsky, Alexey, Stefan Pietschmann, Matthias Niederhausen, and Martin Gaedke (2014). “Towards Awareness and Control in Choreographed User Interface Mashups”. In: *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*. WWW Companion ’14. Seoul, Korea: International World Wide Web Conferences Steering Committee, pp. 389–390.

Tschudnowsky, Alexey, Stefan Pietschmann, Matthias Niederhausen, Michael Hertel, and Martin Gaedke (2014). “From Choreographed to Hybrid User Interface Mashups: A Generic Transformation Approach”. In: *Web Engineering*. Ed. by Sven Casteleyn, Gustavo Rossi, and Marco Winckler. Vol. 8541. Lecture Notes in Computer Science. Springer International Publishing, pp. 145–162.

Wallmüller, E. (2001). *Software-Qualitätsmanagement in der Praxis: Software-Qualität durch Führung und Verbesserung von Software-Prozessen*. Hanser.

Weinhold, Frank, Olexiy Chudnovskyy, Hendrik Gebhardt, and Martin Gaedke (2011). “Geschäftsprozessintegration auf Basis von Telco-Mashups”. In: *INFORMATIK 2011*. Ed. by Pepper Heiß and Schneider Schlingloff. Berlin, Germany: Gesellschaft für Informatik e.V. (GI), p. 376.

Wiedenbeck, Susan (2005). “Facilitators and inhibitors of end-user development by teachers in a school environment”. In: *Proceedings - 2005 IEEE Symposium on Visual Languages and Human-Centric Computing* 2005.1, pp. 215–222.

Wierenga, Berend and Gerrit H. Van Bruggen (2001). “Developing a Customized Decision-support System for Brand Managers”. In: *Interfaces* 31.3 - b, pp. 128–145. URL: `http://dx.doi.org/10.1287/inte.31.4.128.9678`.

Wild, Stefan, Olexiy Chudnovskyy, Sebastian Heil, and Martin Gaedke (2013a). “Customized Views on Profiles in WebID-Based Distributed Social Networks”. In: *Web Engineering*. Ed. by Florian Daniel, Peter Dolog, and Qing Li. Vol. 7977. Lecture Notes in Computer Science. Heidelberg: Springer, pp. 498–501.

Wild, Stefan, Olexiy Chudnovskyy, Sebastian Heil, and Martin Gaedke (2013b). "Protecting User Profile Data in WebID-Based Social Networks Through Fine-Grained Filtering". In: *Current Trends in Web Engineering*. Ed. by Quan Z. Sheng and Jesper Kjeldskov. Vol. 8295. Lecture Notes in Computer Science. Springer, pp. 269–280.

Wild, Stefan, Fabian Wiedemann, Sebastian Heil, Alexey Tschudnowsky, and Martin Gaedke (2015). "ProProtect3: An Approach for Protecting User Profile Data from Disclosure, Tampering, and Improper Use in the Context of WebID". In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems*. Lecture Notes in Computer Science 8990: *Special Issue on Big Data and Open Data XIX*. Ed. by Abdelkader Hameurlain, Josef Küng, Roland Wagner, Devis Bianchini, et al., pp. 87–127.

Wilde, Erik and Martin Gaedke (2008). "Web Engineering Revisited". In: *Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International Academic Conference*. VoCS'08. London, UK: British Computer Society, pp. 41–49.

Wilson, Scott, Florian Daniel, Uwe Jugel, and Stefano Soi (2012). "Orchestrated User Interface Mashups Using W3C Widgets". In: *Current Trends in Web Engineering*. Vol. 7059. LNCS. Springer Berlin Heidelberg, pp. 49–61.

Winer, D. (2003). *RSS 2.0 Specification*. Berkman Center for Internet & Society. URL: `http://cyber.law.harvard.edu/rss/rss.html` (Retrieved Apr. 7, 2012).

Won, Markus, Oliver Stiemerling, and Volker Wulf (2006). "Component-based approaches to tailorable systems". In: *End User Development*, pp. 1–27.

Xiao, Hua, Ying Zou, Ran Tang, Joanna Ng, and Leho Nigul (2010). "A Framework for Automatically Supporting End-Users in Service Composition". In: *The Smart Internet*. Ed. by Mark Chignell, James Cordy, Joanna Ng, and Yelena Yesha. Vol. 6400. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 115–136.

Ye, Yunwen and Gerhard Fischer (2007). "Designing for Participation in Sociotechnical Software Systems". In: *Universal Acess in Human Computer Interaction. Coping with Diversity* 4554, pp. 312–321.

Yeh, Tom, Tsung-Hsiang Chang, and Robert C Miller (2009). “Sikuli: using GUI screenshots for search and automation”. In: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pp. 183–192.

Yu, Jin, Boualem Benatallah, Fabio Casati, and Florian Daniel (2008). “Understanding Mashup Development”. In: *IEEE Internet Computing* 12.5, pp. 44–52.

Zeng, Daniel, Hsinchun Chen, R. Lusch, and Shu-Hsing Li (2010). “Social Media Analytics and Intelligence”. In: *Intelligent Systems, IEEE* 25.6, pp. 13–16.

Schemes

A

A.1 XSD schema of the proposed W3C configuration document extension

Listing A.1: XML Schema Definition (XSD) Schema of the Proposed W3C Configuration Document Extension

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    targetNamespace="http://www.openajax.org/hub">

  <xs:element name="topics">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="topic" maxOccurs="unbounded"
            minOccurs="0">
          <xs:complexType mixed="true">
            <xs:sequence>
              <xs:element type="xs:string" name="description"
                  minOccurs="0" />
              <xs:element type="xs:string" name="schema"
                  minOccurs="0" />
            </xs:sequence>
            <xs:attribute type="xs:anyURI" name="name"
                use="required" />
```

```
          <xs:attribute type="xs:ID" name="id" use="required" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="publications">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="publication" maxOccurs="unbounded"
          minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute type="xs:IDREF" name="topic"
                  use="required" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="subscriptions">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="subscription" maxOccurs="unbounded"
          minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute type="xs:IDREF" name="topic"
                  use="required" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>

</xs:schema>
```

A.2 XSD schema of the proposed OMDL extension

Listing A.2: XSD Schema of the Proposed OMDL Extension

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           attributeFormDefault="unqualified"
                elementFormDefault="qualified"
           targetNamespace="http://www.openajax.org/hub"
           xmlns:oa="http://www.openajax.org/hub">

  <xs:element name="transformations">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transformation">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:anyURI" name="source" />
                <xs:attribute type="xs:anyURI" name="target" />
                <xs:attribute name="type">
                  <xs:annotation>
                    <xs:documentation>
                      javascript,xslt,jsont
                    </xs:documentation>
                  </xs:annotation>
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:enumeration value="javascript" />
                      <xs:enumeration value="xslt" />
                      <xs:enumeration value="jsont" />
                    </xs:restriction>
```

```
                </xs:simpleType>
              </xs:attribute>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="restrictions">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="restriction">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute type="xs:IDREF" name="source"
                  use="required"/>
              <xs:attribute type="xs:IDREF" name="target"
                  use="required"/>
              <xs:attribute type="xs:anyURI" name="topic"
                  use="required"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="isolations">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="isolation">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute type="xs:IDREF" name="source"
                  use="required"/>
```

```
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

</xs:schema>
```

Evaluation Materials

B

B.1 Awareness and Control Facilities

B.1.1 Questionnaire

Pre-Questionnaire

104. On a scale of 1 to 5, how would you describe your English language skills?

105. How would you describe your computer skills? (programmer, expert user, intermediate, basic)

106. Have you ever configured the user interface of a web portal that you use?

107. Do you know what a "widget" is? (yes, no, not sure)

108. Do you know what a "mashup" is? (yes, no, not sure)

Tasks: Test group

A221. Find the CHEAPEST flat in Chemnitz with 3 rooms and a size of 70 to 85 m^2. Afterwards find a similar flat in Dresden with the following characteristics

- same number of rooms
- same level
- similar size ($\pm$ 5m^2)
- similar distance to center ($\pm$ 1km)
- the same extras (plus optional additional ones), e.g., kitchen, parking space, ..

You may use the control mechanisms that have just been described.

Post-Questionnaire: Test group

What is your opinion on the following statements? (strongly agree, agree, undecided, disagree, strongly disagree)

A222. I found the possibility to adjust data exchange useful.

A222. Adjusting the data exchange felt cumbersome.

A223. Do you have additional comments or suggestions regarding the IWC control mechanisms?

Tasks: Control group

B221. Find the CHEAPEST flat in Chemnitz with 3 rooms and a size of 70 to 85 m^2. Afterwards find a similar flat in Dresden with the following characteristics

- same number of rooms
- same level
- similar size (± 5m^2)
- similar distance to center (± 1km)
- the same extras (plus optional additional ones), e.g., kitchen, parking space, ..

You may use the pen and paper if you like.

Post-Questionnaire: Control group

B223. Do you have additional suggestions on how to control data exchange between widgets?

B.1.2 Results

Timestamp	Part. Code		104. English skills	105. Computer skills	106. Portal UI config	107. Knows "widget" term	108. Knows "mashup" term		Group	A221. Time	A222. IWC control useful	A222. IWC control cumbersome	Group	B221. Time
5.8.2013 9:55:18	ANN09DR	PRE	3	expert user	never	Yes.	I'm not	TASKS	TEST	290	5	1		
5.8.2013 11:27:17	FRA03FR	PRE	3	intermediate	never	Yes.	No.	TASKS	TEST	80	5	2		
5.8.2013 13:57:22	KRI19MO	PRE	4	intermediate	never	Yes.	Yes.	TASKS	TEST	370	4	4		
5.13.2013 16:50:27	SAS15BI	PRE	3	intermediate	never	No.	No.	TASKS	TEST	228	5	4		
5.15.2013 14:53:33	AAR05GR	PRE	4	intermediate	never	I'm not	I'm not	TASKS	TEST	76	5	1		
5.22.2013 13:33:59	ARM21OV	PRE	4	intermediate	never	Yes.	Yes.	TASKS	TEST	193	4	3		
5.22.2013 16:04:50	AZI15HA	PRE	4	intermediate	never	Yes.	I'm not	TASKS	TEST	93	3	3		
5.14.2013 17:30:39	CHR18KA	PRE	4	expert user	Start	Yes.	Yes.	TASKS	TEST	141	5	1		
5.15.2013 11:40:13	CHR31SC	PRE	4	expert user	never	Yes.	I'm not	TASKS	TEST	145	4	4		
5.22.2013 11:57:09	JOA02SH	PRE	4	intermediate	never	Yes.	Yes.	TASKS	TEST	112	5	1		
5.22.2013 14:29:56	JOR76NV	PRE	4	intermediate	never	Yes.	No.	TASKS	TEST	116	5	1		
5.14.2013 11:34:55	JUE12JA	PRE	5	programmer	added /	Yes.	Yes.	TASKS	TEST	78	4	1		
5.16.2013 9:48:16	PAU10AL	PRE	3	intermediate	never	I'm not	No.	TASKS	TEST	113	5	1		
5.7.2013 13:53:56	THO25FI	PRE	3	programmer	never	Yes.	I'm not	TASKS	TEST	125	5	2		
5.16.2013 13:43:53	ANJ24KO	PRE	4	intermediate	never	I'm not	No.	TASKS					CONTROL	56
5.7.2013 15:26:57	CAR11BR	PRE	4	intermediate	never	Yes.	I'm not	TASKS					CONTROL	137
5.22.2013 17:00:59	ELS19HJ	PRE	4	intermediate	never	Yes.	No.	TASKS					CONTROL	107
5.15.2013 13:49:42	KAT29SE	PRE	4	intermediate	never	No.	I'm not	TASKS					CONTROL	132
5.13.2013 15:11:07	SUS25PO	PRE	4	expert user	Jira,	Yes.	I'm not	TASKS					CONTROL	162
5.13.2013 11:18:06	CHR18MI	PRE	4	expert user	never	Yes.	Yes.	TASKS					CONTROL	186
5.21.2013 15:16:38	MAR10ME	PRE	4	intermediate	never	I'm not	No.	TASKS					CONTROL	165
5.21.2013 17:45:00	Morgen25	PRE	3	basic	never	No.	No.	TASKS					CONTROL	78
5.21.2013 16:25:09	RAF27RO	PRE	4	intermediate	never	Yes.	No.	TASKS					CONTROL	192
5.7.2013 9:45:59	RAK24	PRE	3	intermediate	never	Yes.	No.	TASKS					CONTROL	102
5.14.2013 13:42:46	STE21BR	PRE	3	programmer	never	Yes.	Yes.	TASKS					CONTROL	355
5.13.2013 9:48:51	THO02MA	PRE	3	expert user	never	Yes.	Yes.	TASKS					CONTROL	347
5.14.2013 15:20:39	ULF16FR	PRE	4	expert user	color,	Yes.	I'm not	TASKS					CONTROL	61

Figure B.1.: Results of Awareness and Control Facilities User Study

B.2 Transformation Editor

B.2.1 Questionnaire

The questionnaire has been translated from german.

Pre-Questionnaire

Name: —

Skills: —(basic, advanced, programmer)

Tasks

The study comprises two tasks. You will be shown a mashup, whose components do not exchange any data. The goal is to create a connection between the components using the given tool.

1. An address book and a map should be combined. Create a connection between the two, so that anytime an address is selected, a corresponding marker on the map appears.

2. An address book and a postcard service should be combined. Establish an appropriate connection. Additionally, a print of "Hallo <First name ><Last name >," should appear on the post card (values in the angle brackets should be replaced by appropriate elements).

Post-Questionnaire

1. I learned the tool quickly (strongly disagree, disagree, agree, strongly agree)

2. It was clear how to solve the task using the tool (strongly disagree, disagree, agree, strongly agree)

3. The creation of transformations should be simpler (strongly disagree, disagree, agree, strongly agree)

There was a situation, where i didn't know how to proceed: —

The following improvements could make the tool more comprehensive: —

Other comments: —

Time for task 1: —

Time for task 2: —

B.2.2 Results

The collected data is shown in Figure B.2.

Date	Part. Code		PC skills		Time Task 1	Time Task 2		Learned Tool quickly	Solution clear	Solution could be easier
09.12.2014	Stefanie	PRE	basic	TASKS	2,6	8,4	POST	3	2	4
09.12.2014	Jürgen	PRE	basic	TASKS	3,4	9,2	POST	4	3	2
03.12.2014	Jens	PRE	average	TASKS	3,1	5,4	POST	3	4	3
04.12.2014	Jessica	PRE	average	TASKS	1,2	7,2	POST	4	3	1
03.12.2014	Michel	PRE	programmer	TASKS	1	6,6	POST	2	2	4
09.12.2014	Philipp	PRE	programmer	TASKS	1	4,1	POST	4	3	3
03.12.2014	Candida	PRE	programmer	TASKS	1,3	6,4	POST	4	2	3

Figure B.2.: Results of Transformation Editor User Study

B.3 Automatic Discovery and Composition Engine

B.3.1 Questionnaire

Pre-Questionnaire

See Appendix B.1.1

Tasks: Test group

Your goal is to build, i.e., populate a workspace with widgets. To this end, please use the "AUTOMATIC COMPOSITION WIDGET" (ACE). It should be already present in the workspace.

A101. Imagine you live in the city of "Dresden" and there is a flood warning. Build a workspace to gather more information on the situation:

1. list messages from your social networks,
2. check where emergency incidents were reported, and
3. compare the flood levels at these locations.

Post-Questionnaire: Test group

What is your opinion on the following statements? (strongly agree, agree, undecided, disagree, strongly disagree)

A102. The Automatic Composer was useful for creating my workspace.

A103. I like the idea of being guided throughout the composition.

A104. I would use the Automatic Composer again, for creating new workspaces.

A105. Do you have further suggestions for improving the Automatic Composer?

Tasks: Control group

Your goal is to build, i.e., populate a workspace with widgets. To this end, please use the Widget Store.

B101. Imagine you live in the city of "Dresden" and there is a flood warning. Build a workspace to gather more information on the situation:

1. list messages from your social networks,
2. check where emergency incidents were reported, and
3. compare the flood levels at these locations.

Post-Questionnaire: Control group

What is your opinion on the following statements? (strongly agree, agree, undecided, disagree, strongly disagree)

B102. It was easy to find the right widgets.

B103. I'm sure that I've found the right widgets.

B104. Do you have further suggestions for improving the Widget Store?

B.3.2 Results

See also Appendix B.1.2 for pre-questionnaire responses. Figure B.3 shows the collected data.

Timestamp	Part. Code	Group	A101. Time	A102. ACE was useful	A103. Like being guided	A104. ACE reuse	Group	B101. Time	B102. Easy to find the right widgets.	B103. Sure that I've found the right widgets.
5.8.2013 9:55:18	ANN09DR	TEST	210	5	4	5				
5.8.2013 11:27:17	FRA03FR	TEST	85	4	5	4				
5.8.2013 13:57:22	KRI19MO	TEST	15	4	3	4				
5.13.2013 16:50:27	SAS15BI	TEST	180	5	5	5				
5.15.2013 14:53:33	AAR05GR	TEST	218	4	4	3				
5.22.2013 13:33:59	ARM21OV	TEST	253	2	3	4				
5.22.2013 16:04:50	AZI15HA	TEST	130	4	4	4				
5.14.2013 17:30:39	CHR18KA	TEST	144	5	5	5				
5.15.2013 11:40:13	CHR31SC	TEST	110	5	5	5				
5.22.2013 11:57:09	JOA02SH	TEST	140	4	4	4				
5.22.2013 14:29:56	JOR76NV	TEST	144	4	4	3				
5.14.2013 11:34:55	JUE12JA	TEST	94	4	5	4				
5.16.2013 9:48:16	PAU10AL	TEST	207	4	2	4				
5.7.2013 13:53:56	THO25FI	TEST	123	4	5	5				
5.16.2013 13:43:53	ANJ24KO						CONTROL	102	4	4
5.7.2013 15:26:57	CAR11BR						CONTROL	175	5	4
5.22.2013 17:00:59	ELS19HJ						CONTROL	88	4	3
5.15.2013 13:49:42	KAT29SE						CONTROL	135	4	5
5.13.2013 15:11:07	SUS25PO						CONTROL	192	4	4
5.13.2013 11:18:06	CHR18MI						CONTROL	300	4	4
5.21.2013 15:16:38	MAR10ME						CONTROL	160	5	4
5.21.2013 17:45:00	Morgen25						CONTROL	159	4	4
5.21.2013 16:25:09	RAF27RO						CONTROL	346	4	2
5.7.2013 9:45:59	RAK24						CONTROL	78	4	3
5.14.2013 13:42:46	STE21BR						CONTROL	112	4	4
5.13.2013 9:48:51	THO02MA						CONTROL	116	4	4
5.14.2013 15:20:39	ULF16FR						CONTROL	145	2	3

Figure B.3.: Results of ADCE User Study

B.4 WebComposition/EUD ICCI Extender

B.4.1 Questionnaire

Pre-Questionnaire

How would you describe your JavaScript skills (1 – no skills, 2 – basic, 3 – advanced, 4 - expert)?

How would you describe your familiarity with Web-based widgets (1 – never heard about, 2 – used in my operating system (e.g. Mac gadgets) or on the Web (e.g. iGoogle), 3 – developed on my own)?

How would you describe your familiarity with W3C widgets (1 – never heard about, 2 – used on the Web, 3 – developed on my own)?

How would you describe your familiarity with Inter-Widget-Communication (1 – never heard about, 2 – have seen it working, 3 – developed on my own)?

How would you describe your familiarity with Open-Ajax Hub (1 – never heard about, 2 – have seen it working, 3 – used during development)?

Tasks

To start, please open the page:

http://localhost:8080/portal

Username: canonical

Password: canonical

Go to the Tab “Movies”.

The mashup consists of three widgets: Movie List, Wikipedia and Youtube. By typing keywords into the widgets, one can search for movies, articles and videos related to the entered keyword. Currently widgets do not communicate with each other, so that input has to be repeated in each of the widgets. Your goal is to make the three widgets communicate. Basic functionality: A keyword typed into the MovieList widget should be passed both to the Wikipedia and the Youtube widgets. Both the widgets should immediately display the search results to the entered keyword. Advanced functionality (optional): if a keyword is entered into Wikipedia widget, it should be propagated to the both other widgets.

Task 1 Perform the extension manually by modifying the source code of the corresponding widgets.

Time for basic functionality: —

Time for advanced functionality (max 15min): —

Advanced functionality solved (yes/no): —

Task 2 Perform the same extension with the help of the IWC-Extender. The IWC-Extender is available at: http://localhost:22222/

Time for basic functionality: —

Time for advanced functionality (max 15min): —

Advanced functionality solved (yes/no): —

Post-Questionnaire

What is your opinion on the following statements? (strongly agree, agree, undecided, disagree, strongly disagree)

1. It was cumbersome to extend widgets with IWC functionality manually

2. The IWC-Extender was easy to use for extending the given widgets.

3. I learned the tool quickly

4. As for my experience with widgets, the tool would be well applicable to a wide range of different widgets and scenarios

5. I would use the IWC-Extender again for adjusting other widgets.

Do you have suggestions for improving the IWC-Extender?

Which steps or actions during manual extension of widgets towards IWC do you think are the most cumbersome and have to be supported/automated by tools?

B.4.2 Results

Date	Part. Code		Javascript skills	Familiarity with Web-based Widgets	Familiarity with W3C Widgets	Familiarity with IWC	Familiarity with OpenAjaxHub		Time Basic (sec)	Advanced solved within 15 min	Time Basic with Tool (sec)	Time Advanced solved with Tool	Time Advanced with Tool		A1. Manually is cumbersome	A2. Extender is easy to use	A3. Learned quickly	A4. Well applicable	A5. Would reuse
09.12.2014	AND14SE	PRE	2	1	1	2	2	TASKS	1625	0	1346	1	889	POST	4	4	4	4	5
09.12.2014	PHI04AV	PRE	3	3	2	3	2	TASKS	807	1	164	1	120	POST	4	4	4	4	5
03.12.2014	HAI12RE	PRE	2	3	3	2	1	TASKS	3935	0	215	1	215	POST	2	4	4	4	4
04.12.2014	MIC08HE	PRE	3	3	2	3	2	TASKS	840	1	100	1	130	POST	4	5	4	3	4
03.12.2014	SEB17JO	PRE	3	3	2	3	3	TASKS	1372	0	403	1	133	POST	4	4	5	4	4
09.12.2014	ALE01AL	PRE	2	2	1	2	1	TASKS	1301	0	249	1	205	POST	4	4	4	2	3
03.12.2014	FAB01PU	PRE	3	3	2	3	2	TASKS	1220	1	368	1	125	POST	3	5	5	2	3
10.12.2014	MAR17RE	PRE	3	2	2	2	1	TASKS	909	1	330	1	52	POST	3	4	5	4	5
10.12.2014	MIC18KA	PRE	3	3	3	3	3	TASKS	881	1	138	1	108	POST	4	5	5	4	5
10.12.2014	TOB06HA	PRE	2	2	2	2	1	TASKS	1915	0	425	1	115	POST	5	5	5	4	5

Figure B.4.: Results of WebComposition/IE User Study

B.5 Performance Evaluation of the WebComposition/EUD Artifact Library

Table B.1.: Median Response Times of WebComposition/EUD-AL in Seconds (10 Runs for Each Test)

Test / Number of components	10	100	1000	5000	10000	50000	100000	200000	250000	268242
SPARQL request using *contains*	0.036	0.031	0.093	0.328	0.608	2.792	5.569	10.109	11.919	12.651
SPARQL request using *regex*	0.036	0.046	0.093	0.312	0.577	2.793	5.600	10.249	12.137	12.839
SPARQL request using *search:text*	0.039	0.031	0.032	0.047	0.047	0.047	0.047	0.031	0.047	0.047
SPARQL request for superclass	0.036	0.046	0.203	0.905	1.179	9.048	18.112	23.104	30.530	33.041
SPARQL request for subclass 1 (Netvibes)	0.036	0.046	0.187	0.779	1.529	7.683	15.397	11.388	15.132	16.583
SPARQL request for subclass 2 (W3C)	0.036	0.031	0.187	0.779	1.529	7.675	15.366	11.450	15.210	16.552
Publishing of one component	0.249	0.234	0.141	0.125	0.109	0.109	0.203	0.265	0.608	0.593
Accessing a random component	0.036	0.046	0.032	0.062	0.062	0.188	0.358	0.656	0.842	0.889

Figures

2.1 FireView Dashboard: A Software Solution for Fire and Emergency Response Agencies 10
2.2 JReport Dashboards: A Web Application for Visualization of KPIs 12
2.3 Immobilienscout24: A Web Application for Real Estate Management 13

3.1 Exploratory and Evolutionary Development of Software by Scientists 25
3.2 Adaptation of Waterfall Model Towards Component-Based Development 29
3.3 The Y model for Component-Based Software Development 31
3.4 The W model for Component-Based Software Development 32
3.5 Evolution of Web Applications Based on WebComposition Process Model 33
3.6 Lifecycle Model of Web Mashups 34
3.7 Model-Driven Development Process 36
3.8 WebML Development Process 38
3.9 End-User-Friendly Dashboard for Administration of WCMS 42
3.10 Example of a Dashboard Implemented Using Liferay . . . 46
3.11 Specification of Report Layout in JReport 47
3.12 A Platform for End-User Tailoring of Component-Based Software 50
3.13 Yahoo Pipes: a Data Mashup Editor 54

3.14 ServFace Builder: a WYSIWYG Service Composition environment 55
3.15 PEUDOM: a Visual Environment for Development of UI Mashups 57

4.1 Quality-ensuring Design Measures of the WebComposition/EUD Approach 64
4.2 WebComposition/EUD Component Model 68
4.3 WebComposition/EUD Composition Model 72
4.4 Example DSS for Flood Catastrophe Management 75
4.5 WebComposition/EUD Role Model 78
4.6 WebComposition/EUD Process Model 79
4.7 WebComposition/EUD Toolkit and Corresponding Target Roles 87

5.1 Conceptual Architecture of the WebComposition/EUD Composition Platform 92
5.2 Example of a W3C Packaged App Content 94
5.3 Extensions of W3C Configuration Document to Describe ICC Behavior 95
5.4 OMDL Extensions to Describe ICC Configuration 100
5.5 Implementation of the Composition Platform 103
5.6 Awareness and Control Facilities 108
5.7 Visual Definition of Transformation Functions 109
5.8 Evaluation Application for Testing Usability of Awareness and Control Facilities 112
5.9 Impact of Awareness and Control Facilities on Efficiency of Composition Developers 114
5.10 Usability Evaluation of Awareness and Control Facilities . 114
5.11 Tasks for Evaluation of Transformations Editor 116
5.12 Usability Evaluation of the Transformations Editor 118

6.1 Discovery and Composition Process 123
6.2 Excerpt from the ADCE Domain Ontology 124
6.3 Excerpt from the ADCE Goal Ontology 125
6.4 Dialog-based Solution Discovery and Composition 128
6.5 Acceleration of Mashup Creation 131

6.6 Acceptance of the Guidance Idea 132
6.7 Perceived Usefulness (Left) and Reuse Willingness (Right) of ADCE 132
6.8 Observation and Automation of Repeated Input 150

7.1 Conversion of Proprietary Widgets into WebComposition/EUD Components 161
7.2 Metamodel WebComposition/EUD Component Packages (based on W3C Packaged Web Apps) 162
7.3 Success Rate of WebComposition/EUD-CC 168
7.4 Interactive Demonstration of Desired ICC Functionality . . 174
7.5 Architecture of the WebComposition/EUD-IE 176
7.6 WebComposition/EUD-IE: Evaluation tasks 181
7.7 WebComposition/EUD-IE: Performance and Success Rate Increase 181
7.8 WebComposition/EUD-IE: Ease of Use and Learnability . . 182
7.9 WebComposition/EUD-IE: Applicability and Reuse Willingness 183
7.10 Architecture of the WebComposition/EUD-AL 187
7.11 Processing of SPARQL Templates 193
7.12 Response Times of DGS for Component Search Queries by Keyword 198
7.13 Response Times of DGS for Component Search Queries by Component Type 199
7.14 Response Times of DGS for Adding and Accessing New Components 200

8.1 Application of the WebComposition/EUD Framework to build a Public Information Screen 208
8.2 Application of the WebComposition/EUD Framework to Travel Planning Domain 209
8.3 Application of the WebComposition/EUD Framework to Implement Telecommunication Dashboards 211

B.1 Results of Awareness and Control Facilities User Study . . 258
B.2 Results of Transformation Editor User Study 261
B.3 Results of ADCE User Study 264

B.4 Results of WebComposition/IE User Study 268

Tables

3.1 Comparison of Analyzed State-of-the-Art Technologies . . 60

8.1 Comparison of the WebComposition/EUD Framework with State-of-the-Art Technologies 205

B.1 Median Response Times of WebComposition/EUD-AL in Seconds (10 Runs for Each Test) 269

Listings

5.1 Inter-Component Communication using OpenAjax Hub . 96
5.2 Extension of the W3C Configuration Document 97
5.3 Topic Declaration using a JSON-Scheme Document . . . 98
5.4 OMDL Description of the Emergency Response Application 99
5.5 Example of Proposed OMDL Extension to Describe ICC Configuration 102

7.1 Definition of XQuery Templates 189
7.2 Automatic Extraction and Publishing of Packaged Web App Metadata 191
7.3 Automatic Extraction and Publishing of Image Metadata 192
7.4 Example of Access Control Rules Definition in Data Grid Service 195

A.1 XSD Schema of the Proposed W3C Configuration Document Extension 249
A.2 XSD Schema of the Proposed OMDL Extension 251

Doctoral Dissertations in Web Engineering and Web Science

(1) Heinrich, Matthias (2014)
Enriching Web Applications Efficiently with Real-Time Collaboration Capabilities
ISBN 978-3-941003-25-9
Volltext: `http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-149948`

(2) Speicher, Maximilian (2016)
Search Interaction Optimization: A Human-Centered Design Approach
ISBN 978-3-944640-99-0
Volltext: `http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-208102`

(3) Wild, Stefan (2017)
Enhancing Security in Managing Personal Data by Web Systems
ISBN 978-3-96100-010-4
Volltext: `http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-217284`

(4) Tschudnowsky, Alexey (2017)
End-User Development of Web-based Decision Support Systems
ISBN 978-3-96100-014-2
Volltext: `http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-21982`