Complexity Management in Graphical Models

DOCTORAL THESIS

FOR THE DEGREE OF A
DOCTOR OF INFORMATICS

AT THE FACULTY OF ECONOMICS,
BUSINESS ADMINISTRATION AND
INFORMATION TECHNOLOGY
OF THE
UNIVERSITY OF ZURICH

by
TOBIAS REINHARD
from
Horw LU

Accepted on the recommendation of

PROF. DR. M. GLINZ
PROF. DR. H. C. GALL

2010

The Faculty of Economics, Business Administration and Information Technology of the University of Zurich herewith permits the publication of the aforementioned dissertation without expressing any opinion on the views contained therein.

Zurich, October 27, 2010[1]

The Vice Dean of the Academic Program in Informatics: Prof. Dr. H. C. Gall

[1] Date of the graduation

Berichte aus der Softwaretechnik

Tobias Reinhard

Complexity Management in Graphical Models

Shaker Verlag
Aachen 2011

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: Zürich, Univ., Diss., 2010

Copyright Shaker Verlag 2011
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8322-9851-7
ISSN 1433-9986

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

Graphical or visual representations play a central role in the software life cycle as a means to make the immaterial software more tangible and accessible. While such drawings or diagrams facilitate a “computational offloading” when reasoning about a system, the complexity of today’s software systems makes them often extremely big and cluttered. One way to cope with this size and complexity is to use hierarchical and aspectual decompositions to split the models into manageable and understandable parts. Such a decomposition mechanism is the basic idea behind the ADORA approach: It uses an integrated, inherently hierarchical model together with a tool that generates abstractions in the form of diagrams of manageable complexity. The underlying complexity management mechanism combines two concepts: (i) a fisheye zoom visualization which shows local detail and its surrounding global context in one single view and (ii) a dynamic generation of different views by filtering specific model elements.

The work at hand covers the technical foundations of this complexity management mechanism. While the simplicity of the basic concept contributes largely to its appeal, the actual realization in a computer-based tool has to cope with a lot of conceptual and technical problems and trade-offs. Besides the presentation and discussion of the actual data structures and algorithms, the detailed requirements they have to fulfill are covered as well. An improved fisheye zoom algorithm that employs the concept of interval scaling and solves the problem of having a user-editable layout which is stable under multiple zoom operations builds the basis for the dynamic adaption of a diagram. This algorithm can be extended to adapt the layout if model elements are filtered to generate different views on the model. Additionally, it can be used to support the model editing by adapting the layout automatically. Since these automatic layout adjustments result in a dynamic, constantly changing diagram, the links or lines connecting model elements have to be adapted, too. As a solution to that problem, an automatic line routing algorithm that produces an aesthetically appealing layout and routes in real time has been developed. The basic data structure of this algorithm can also be used to automatically place the labels accompanying the links.

Zusammenfassung

Graphische Repräsentationen spielen im Software Lebenszyklus eine zentrale Rolle dabei, immaterielle Software fassbarer und zugänglicher zu machen. Obwohl solche Grafiken oder Diagramme das sogenannte "computational offloading" beim Verstehen eines Systems fördern, führt die Komplexität heutiger Softwaresysteme oftmals zu sehr grossen und überladenen Modellen. Eine Möglichkeit um dieser Grösse und Komplexität Herr zu werden liegt in einer hierarchischen und aspekt-basierten Dekomposition des Modells in handhabbare und verständliche Teile. Eine solche Dekomposition ist die grundlegende Idee hinter ADORA: Der Ansatz beruht auf einem integrierten, inhärent hierarchischen Modell zusammen mit einem Werkzeug, welches Abstraktionen in der Form von Diagrammen handhabbarer Grösse und Komplexität generiert. Der Mechanismus zur Handhabung der Komplexität kombiniert zwei Konzepte: a) eine sogenannte Fischaugen-Visualisierung, welche lokale Details zusammen mit dem umgebenden Kontext in einer einzigen Ansicht darstellt, und b) eine dynamische Generierung verschiedener Sichten durch das Ausblenden spezifischer Modellelemente.

Die vorliegende Arbeit beschäftigt sich mit den technischen Grundlagen dieses Ansatzes des Umgangs mit Komplexität. Obwohl die Einfachheit des Konzepts einen grossen Teil dessen Attraktivität ausmacht, tauchen bei der Realisierung in einem rechnergestützten Werkzeug eine ganze Menge technischer Probleme und Zielkonflikte auf. Neben der Präsentation und Diskussion der dafür benötigten Datenstrukturen und Algorithmen, werden auch die Anforderungen, welche diese zu erfüllen haben, behandelt. Ein verbesserter Fischaugen Zoomalgorithmus, der auf dem Prinzip der Skalierung von Intervallen beruht und sowohl die Editierbarkeit des Layouts als auch dessen Stabilität bei mehreren Zoomoperationen garantiert, bildet die Basis für die dynamische Anpassung des Layouts. Dieser Algorithmus kann, mit kleinen Anpassungen, auch dazu verwendet werden, das Layout beim Generieren verschiedener Sichten durch das Ausblenden spezifischer Modellelemente anzupassen. Zusätzlich unterstützt er den Benutzer beim Editieren des Modells durch eine automatische Anpassung. Da diese automatischen Veränderungen des Layouts in dynamischen, sich ständig ändernden Diagrammen resultieren, müssen auch die Linien, welche die Modellelemente verbinden, angepasst werden. Als Lösung für dieses Problem wurde ein automatischer Linienführungsalgorithmus entwickelt, der in Echtzeit ästhetisch ansprechende Linien generiert. Die Datenstruktur dieses Algorithmus kann zusätzlich auch zur automatischen Platzierung von Linienbeschriftungen verwendet werden.

Contents

List of Figures **xiii**

I Introduction and Background **1**

1 Introduction **3**

1.1 Motivation 3

1.2 Goals and Contributions 4

1.3 Thesis Outline 6

2 Graphical Modeling **9**

2.1 Models and Their Representation 9

2.2 Graphical Models in Software Engineering 14

2.3 Tool Support for the Modeling Process 21

2.4 Aesthetics in Diagrams 24

2.5 Secondary Notation and Mental Map 25

3 Complexity of Graphical Models **27**

3.1 Flat Models 28

3.2 Horizontal Abstraction 35

3.3 Vertical Abstraction 36

4 ADORA 45

4.1 The ADORA Language 45

4.2 The ADORA Tool 49

II Fisheye Zooming 51

5 Desired Properties of a Zoom Algorithm 53

5.1 Compact Layout 53

5.2 Disjoint Nodes 54

5.3 Preserve the Mental Map 55

5.4 Layout Stability 58

5.5 Permit Editing Operations 59

5.6 Runtime 60

5.7 Multiple Focal Points 60

5.8 Smooth Transitions 61

5.9 Small Interaction Overhead 61

5.10 Minimal Node Size 62

6 Existing Fisheye Zoom Techniques 63

6.1 The Force-Scan Algorithm 64

6.2 SHriMP 65

6.3 Berner 70

6.4 The Continuous Zoom 71

7 Zoom Algorithm 77

7.1 Data Structure 78

7.2 Zoom Operations 80

7.3 Bottom-up Zooming 87

8 Dynamically Generated Views 89

8.1 Node Filters 90

8.2 Filter Operation 90

9 Editing Support 95

9.1 Inserting Nodes 96

9.2 Removing Nodes 101

9.3 Changing the Bounds of a Node 103

9.4 Compacting Nodes 103

10 Discussion of the Zoom Algorithm 105

10.1 Properties of the Zoom Algorithm 105

10.2 Algorithmic Complexity 116

III Line Routing 117

11 Lines in Graphical Models 119

11.1 Lines in Hierarchical Models 119

11.2 Lines in a Dynamic Layout 122

11.3 Desired Properties of a Line Routing Algorithm 123

12 Existing Line Routing Approaches 129

12.1 Lines as Part of the Automatic Graph Drawing 129

12.2 Routing on the Visibility Graph 130

12.3 Grid-Based Routing 131

12.4 Routing in Incremental Layouts 133

13 Tile Maze Router 135

13.1 Data Structure 135

13.2 Channel Finding 141

13.3 Bend Points Calculation 143

13.4 Line Crossings 147

13.5 Calculation of the Start and End Point 149

13.6 Reflexive Lines 152

13.7 Discussion of the Routing Algorithm 152

14 Line Labels 159

14.1 Label Placement Rules 160

14.2 Basic Approaches 161

14.3 Tile-based Label Placement 163

IV Validation 167

15 Constructive Validation 169

15.1 Layout Functionality 169

15.2 Technical Aspects 174

16 Experimental Validation 175

16.1 Experiment 175

16.2 Results 179

16.3 Validity of the Experiment 183

V Conclusions 185

17 Conclusions and Future Work 187

17.1 Summary and Achievements 187

17.2 Limitations 189

17.3 Future Work 189

VI Appendix 191

A Details of the Experimental Validation 193

A.1 Case Studies 193

A.2 Questionnaires 196

A.3 Raw Results 202

A.4 Statistical Analysis 205

Bibliography 207

List of Figures

1.1 Typical layout problems resulting from only rudimentary tool support 6

1.2 Enhanced tool support to automate layouting tasks 7

2.1 Hierarchy of UML 2.0 diagrams 18

3.1 Graphical fisheye view of a simple software system model 34

3.2 Saul Steinberg's "View of the World from the 9th Avenue" 40

3.3 Generalized Fisheye View of a simple tree 42

3.4 Fisheye zooming 44

5.1 Produce a compact layout 54

5.2 Dual Graph 57

5.3 Layout stability 58

6.1 The force-scan algorithm 64

6.2 Partitions to determine the translation vectors 66

6.3 Proximity preservation strategy 67

6.4 Alternative proximity preservation strategy 68

6.5 Overlapping nodes in SHriMP . . . 69

6.6 Calculation of the translation vectors in Berner's algorithm . . . 70

6.7 Overlapping nodes in Berner's algorithm . . . 71

6.8 Normal geometry of the Continuous Zoom . . . 72

6.9 Node A has been scaled down by factor 0.5 . . . 74

7.1 Interval structure . . . 78

7.2 Zooming-in node A . . . 82

7.3 Zooming-out node A . . . 85

7.4 Bottom-up zooming in the hierarchy . . . 87

8.1 Filtering nodes C, D and E . . . 91

8.2 Minimal size of node A if node B is hidden . . . 93

9.1 Inserting a new node . . . 96

9.2 Situation after the expanded node D' has been inserted . . . 97

9.3 Final situation after node D has been inserted . . . 98

9.4 Calculation of the maximal available free space . . . 99

9.5 Removing node B from the diagram . . . 102

9.6 Final situation after node B has been removed . . . 102

10.1 Nodes B and C are shadowed by node A . . . 106

10.2 Nodes A, B and C are always disjoint after a zoom operation . . . 107

10.3 The vertical orthogonal ordering between nodes A and D is not preserved . . . 109

10.4 Maintain the proximity relations when node A is zoomed-out . . . 110

10.5 Maintain the stability of the layout irrespective of the order of zoom operations 111

10.6 Relation between the insert and remove operation . . . 112

10.7 Layout is no longer stable after an editing operation . . . 114

10.8 Number of intervals in the interval structure . . . 116

11.1 Nodes C, D, G and I are potential obstacles for the link between F and J . . . 120
11.2 Abstract relationship between nodes B and J after B has been zoomed-out . . . 121
11.3 Direct line between nodes A and C after node B has been zoomed-out 123
11.4 Line routing and the mental map 126
11.5 Secondary notation employed in a line 127

12.1 Visibility graph 130
12.2 Lee's algorithm 132
12.3 Rectangulation 133

13.1 Corner stitching structure 136
13.2 Locating the tile that contains point p 138
13.3 Inserting a tile into the corner stitching structure 139
13.4 Finding a path of space tiles from s to t 142
13.5 Possible constellation for the orthogonal routing 144
13.6 Calculate the bending points for an orthogonal line 146
13.7 Choose among multiple possible solutions 147
13.8 Weighted tiles 148
13.9 Calculating the position of the start and end point 150
13.10 Routing of a reflexive line 152
13.11 Additional tiles for a line segment 155
13.12 Additional line tiles for a node 155
13.13 Additional line tiles for a bend 156
13.14 Additional line tiles for line crossing 156

14.1 Overlapping label after zooming-out node A 160
14.2 Move nodes A and B to provide the required space to the label 161
14.3 Reroute the line to provide the required space to the label 162
14.4 Calculation of the label space on the corner stitching structure 164

14.5 Candidate label positions 165
14.6 Solution for Fig. 14.1 166

15.1 Base and structural view of the elevator system 170
15.2 Behavior view with focus on the UpButton 171
15.3 Inserting a new node 172
15.4 Automatic line routing and label placement 173

16.1 Screenshot of the empirical testing environment 178
16.2 Performance points for the mine drainage system 180
16.3 Performance points for the elevator system 181
16.4 Interaction of Subject 1 with the mine drainage system 182

A.1 Mine drainage system 194
A.2 Elevator system 195