Ana Moreira
John Grundy (Eds.)

Early Aspects: Current Challenges and Future Directions

LNCS 4765

10th International Workshop
Vancouver, Canada, March 2007
Revised Selected Papers

Springer

Ana Moreira John Grundy (Eds.)

Early Aspects: Current Challenges and Future Directions

10th International Workshop
Vancouver, Canada, March 13, 2007
Revised Selected Papers

Springer

Volume Editors

Ana Moreira
Universidade Nova de Lisboa
Faculdade de Ciências e tecnologia
Departamento de Informaática
2829-516 Caparica, Portugal
E-mail: amm@di.fct.unl.pt

John Grundy
University of Auckland
Department of Electrical and Computer Engineering and
Department of Computer Science
Private Bag 92019, Mail Centre, Auckland, 1142, New Zealand
E-mail: john-g@cs.auckland.ac.nz

Library of Congress Control Number: 2007941794

CR Subject Classification (1998): D.2, D.3, I.6, H.4, K.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-76810-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-76810-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12192458 06/3180 5 4 3 2 1 0

Lecture Notes in Computer Science 4765

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Preface

Celebrating Five Years of Early Aspects

The early aspects community had its origins in the "Early Aspects: Requirements Engineering and Architecture Design" workshop organized during the first international conference on Aspect-Oriented Software Development (AOSD), in March 2002. Since then, the early aspects community has grown rapidly. At the time this project started, the Early Aspects Steering Committee (www.early-aspects.net) had organized nine editions of the Early Aspects workshop in conferences such as AOSD OOPSLA, ICSE and SPLC and edited two special issues in international journals. Workshop attendance has exceeded 200, and from these more than 60% were different individuals. This number corresponds to just over 20 participants per workshop, despite the fact that participation was allowed only to authors of accepted papers or invited researchers.

However, the early aspects community is much larger than that. A considerable number of papers have been published regularly in journals, books and conferences where the early aspects workshop has not yet been organized. The number and range of submissions to the workshop series have demonstrated that the field has a solid base of continuous research being done by established groups around the world.

The early-aspects community is now self-sustaining and continuously expanding. Therefore, we felt that the fifth anniversary of the first early aspects workshop was an appropriate juncture to upgrade the autonomous standing of the community by providing it with its own formal publication. In this way, relevant new work can be showcased in a dedicated publication, instead of being dispersed across several different events with more informal proceedings.

What Are Early Aspects?

Traditionally, aspect-oriented software development (AOSD) has focused on the implementation phase of the software lifecycle: aspects are identified and captured mainly in code. Therefore, most current AOSD approaches place the burden for aspect identification and management on the programmer working at low levels of abstraction. However, aspects are often present well before the implementation phase, such as in domain models, requirements and software architecture.

Identification and capture of these early aspects ensure that aspects related to the problem domain (as opposed to merely the implementation) will be appropriately captured, reasoned about and available. This offers improved opportunities for early recognition and negotiation of trade-offs and allows forward and backward aspect traceability. This makes requirements, architecture, and implementation more seamless, and allows a more systematic application of aspects.

Early aspects are crosscutting concerns that exist in the early life cycle phases of software development, including requirements analysis, domain analysis and architecture design activities. Early aspects cannot be localized using traditional software develop-

ment divide-and-conquer functional decomposition techniques and tend to be scattered over multiple early life cycle modules. This reduces modularity and reusability, potentially leading to serious maintenance and evolution problems.

Whereas conventional aspect-oriented software development approaches are mainly concerned with identifying aspects at the programming level, early aspects work focuses on the impact of crosscutting concerns during earlier activities of software development. Identifying, modularizing and managing aspects early in the software development process has a large and positive impact on the whole software system.

About This Volume

The tenth edition of the Early Aspects workshop was a success. The quality of the technical program, marked by three special moments, attracted some 50 participants in all. We began with a brilliant keynote given by Anthony Finkelstein on "Aspects, Views and Processes" where he emphasized the need to step away from considering aspects purely in terms of representation and start exploring the tangled relationship between processes and aspects.

The International Programme Committee selected ten high-quality papers for presetnation at the workshop. These were presented in four single-stream sessions with a discussant and emphasis on audience participation and debate. In a second, post-workshop phase, the authors were requested to produce a new version of their work to address the comments received from PC members and also specific points that were raised by their paper discussant and other participants during the workshop. These revised papers were submitted to a subset of the original PC members for a second review, each paper in this volume thus going through a two-stage revision process.

Finally a panel entitled "Early Aspects: Are There Any Other Kind?" was run at the conclusion of the workshop. The panel leader was Awais Rashid (University of Lancaster) and the panelists were Anthony Finkelstein (University College London), Gregor Kiczales (University of British Columbia), Maja D'Hondt (INRIA) and Ana Moreira (Universidade Nova de Lisboa). This panel attracted to the room not only early aspects researchers but also a whole set of software engineers usually more interested in later activities of software development. A summary of the discussions closes this volume with an editorial authored by Awais Rashid.

We would like to thank all the Programme Committee members for evaluating the papers, the authors for submitting their work and for improving it according to comments received from reviewers and discussants, and the AOSD-Europe project for sponsoring the workshop. We also would like to thank the papers discussants: Paul Clements, Ruzanna Chitchyan, Monica Pinto and Bedir Tekinerdogan. A special word of thanks is due to Anthony Finkelstein for his superb keynote and to Awais Rashid for making the panel a success. Many thanks to Gregor, Anthony, Maja and Ana for providing the participants with an enthusiastic and lively discussion. No doubt remains about the fundamental role played by early aspects in aspect-oriented software development.

September 2007

Ana Moreira
John Grundy

Organization

Program Committee

Alessandro Garcia (University of Lancaster)
Anthony Finkelstein (University College London)
Awais Rashid (University of Lancaster)
Bashar Nuseibeh (Open University)
Bedir Tekinerdogan (University of Twente)
Charles Haley (Open University)
Christa Schwanninger (Siemens)
Dominik Stein (University of Essen)
Elisa Baniassad (Univerity of Hong Kong)
Jaelson Castro (University of Pernanbuco)
Jean-Marc Jezequel (IRISA)
Jeff Gray (University of Alabama at Birmingham)
João Araújo (Universidade Nova de Lisboa)
John Hosking (University of Auckland)
Jon Whittle (George Mason University)
Juan Hernandez (University of Extremadura)
Julio Leite (PUC, Brazil)
Krzysztof Czarnecki (University of Waterloo)
Len Bass (Carnegie Mellon University)
Lidia Fuentes (University of Malaga)
Michael Jackson (Open University)
Oscar Pastor (University of Valencia)
Paul Clements (SEI, USA)
Robert Walker (University of Calgary)
Ruzanna Chitchyan (Lancaster University)
Siobhan Clarke (Trinity College Dublin)
Stan Sutton Jr. (IBM T. J. Watson Research Center)
Stefan Hanenberg (University of Essen)

Lecture Notes in Computer Science

Sublibrary 2: Programming and Software Engineering

For information about Vols. 1– 4257
please contact your bookseller or Springer

Vol. 4902: P. Hudak, D.S. Warren (Eds.), Practical Aspects of Declarative Languages. X, 333 pages. 2007.

Vol. 4888: F. Kordon, O. Sokolsky (Eds.), Composition of Embedded Systems. XII, 221 pages. 2007.

Vol. 4849: M. Winckler, H. Johnson, P. Palanque (Eds.), Task Models and Diagrams for User Interface Design. XIII, 299 pages. 2007.

Vol. 4839: O. Sokolsky, S. Taşıran (Eds.), Runtime Verification. VI, 215 pages. 2007.

Vol. 4834: R. Cerqueira, R.H. Campbell (Eds.), Middleware 2007. XIII, 451 pages. 2007.

Vol. 4829: M. Lumpe, W. Vanderperren (Eds.), Software Composition. VIII, 281 pages. 2007.

Vol. 4824: A. Paschke, Y. Biletskiy (Eds.), Advances in Rule Interchange and Applications. XIII, 243 pages. 2007.

Vol. 4807: Z. Shao (Ed.), Programming Languages and Systems. XI, 431 pages. 2007.

Vol. 4799: A. Holzinger (Ed.), HCI and Usability for Medicine and Health Care. XVI, 458 pages. 2007.

Vol. 4789: M. Butler, M.G. Hinchey, M.M. Larrondo-Petrie (Eds.), Formal Methods and Software Engineering. VIII, 387 pages. 2007.

Vol. 4767: F. Arbab, M. Sirjani (Eds.), International Symposium on Fundamentals of Software Engineering. XIII, 450 pages. 2007.

Vol. 4765: A. Moreira, J. Grundy (Eds.), Early Aspects: Current Challenges and Future Directions. X, 199 pages. 2007.

Vol. 4764: P. Abrahamsson, N. Baddoo, T. Margaria, R. Messnarz (Eds.), Software Process Improvement. XI, 225 pages. 2007.

Vol. 4762: K.S. Namjoshi, T. Yoneda, T. Higashino, Y. Okamura (Eds.), Automated Technology for Verification and Analysis. XIV, 566 pages. 2007.

Vol. 4758: F. Oquendo (Ed.), Software Architecture. XVI, 340 pages. 2007.

Vol. 4757: F. Cappello, T. Herault, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. XVI, 396 pages. 2007.

Vol. 4753: E. Duval, R. Klamma, M. Wolpers (Eds.), Creating New Learning Experiences on a Global Scale. XII, 518 pages. 2007.

Vol. 4749: B.J. Krämer, K.-J. Lin, P. Narasimhan (Eds.), Service-Oriented Computing – ICSOC 2007. XIX, 629 pages. 2007.

Vol. 4748: K. Wolter (Ed.), Formal Methods and Stochastic Models for Performance Evaluation. X, 301 pages. 2007.

Vol. 4741: C. Bessière (Ed.), Principles and Practice of Constraint Programming – CP 2007. XV, 890 pages. 2007.

Vol. 4735: G. Engels, B. Opdyke, D.C. Schmidt, F. Weil (Eds.), Model Driven Engineering Languages and Systems. XV, 698 pages. 2007.

Vol. 4716: B. Meyer, M. Joseph (Eds.), Software Engineering Approaches for Offshore and Outsourced Development. X, 201 pages. 2007.

Vol. 4709: F.S. de Boer, M.M. Bonsangue, S. Graf, W.-P. de Roever (Eds.), Formal Methods for Components and Objects. VIII, 297 pages. 2007.

Vol. 4680: F. Saglietti, N. Oster (Eds.), Computer Safety, Reliability, and Security. XV, 548 pages. 2007.

Vol. 4670: V. Dahl, I. Niemelä (Eds.), Logic Programming. XII, 470 pages. 2007.

Vol. 4652: D. Georgakopoulos, N. Ritter, B. Benatallah, C. Zirpins, G. Feuerlicht, M. Schoenherr, H.R. Motahari-Nezhad (Eds.), Service-Oriented Computing ICSOC 2006. XVI, 201 pages. 2007.

Vol. 4640: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development IV. IX, 191 pages. 2007.

Vol. 4634: H. Riis Nielson, G. Filé (Eds.), Static Analysis. XI, 469 pages. 2007.

Vol. 4620: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development III. IX, 201 pages. 2007.

Vol. 4615: R. de Lemos, C. Gacek, A. Romanovsky (Eds.), Architecting Dependable Systems IV. XIV, 435 pages. 2007.

Vol. 4610: B. Xiao, L.T. Yang, J. Ma, C. Muller-Schloer, Y. Hua (Eds.), Autonomic and Trusted Computing. XVIII, 571 pages. 2007.

Vol. 4609: E. Ernst (Ed.), ECOOP 2007 – Object-Oriented Programming. XIII, 625 pages. 2007.

Vol. 4608: H.W. Schmidt, I. Crnković, G.T. Heineman, J.A. Stafford (Eds.), Component-Based Software Engineering. XII, 283 pages. 2007.

Vol. 4591: J. Davies, J. Gibbons (Eds.), Integrated Formal Methods. IX, 660 pages. 2007.

Vol. 4589: J. Münch, P. Abrahamsson (Eds.), Product-Focused Software Process Improvement. XII, 414 pages. 2007.

Vol. 4574: J. Derrick, J. Vain (Eds.), Formal Techniques for Networked and Distributed Systems – FORTE 2007. XI, 375 pages. 2007.

Vol. 4556: C. Stephanidis (Ed.), Universal Access in Human-Computer Interaction, Part III. XXII, 1020 pages. 2007.

Vol. 4555: C. Stephanidis (Ed.), Universal Access in Human-Computer Interaction, Part II. XXII, 1066 pages. 2007.

Vol. 4554: C. Stephanidis (Ed.), Universal Acess in Human Computer Interaction, Part I. XXII, 1054 pages. 2007.

Vol. 4553: J.A. Jacko (Ed.), Human-Computer Interaction, Part IV. XXIV, 1225 pages. 2007.

Vol. 4552: J.A. Jacko (Ed.), Human-Computer Interaction, Part III. XXI, 1038 pages. 2007.

Vol. 4551: J.A. Jacko (Ed.), Human-Computer Interaction, Part II. XXIII, 1253 pages. 2007.

Vol. 4550: J.A. Jacko (Ed.), Human-Computer Interaction, Part I. XXIII, 1240 pages. 2007.

Vol. 4542: P. Sawyer, B. Paech, P. Heymans (Eds.), Requirements Engineering: Foundation for Software Quality. IX, 384 pages. 2007.

Vol. 4536: G. Concas, E. Damiani, M. Scotto, G. Succi (Eds.), Agile Processes in Software Engineering and Extreme Programming. XV, 276 pages. 2007.

Vol. 4530: D.H. Akehurst, R. Vogel, R.F. Paige (Eds.), Model Driven Architecture - Foundations and Applications. X, 219 pages. 2007.

Vol. 4523: Y.-H. Lee, H.-N. Kim, J. Kim, Y.W. Park, L.T. Yang, S.W. Kim (Eds.), Embedded Software and Systems. XIX, 829 pages. 2007.

Vol. 4498: N. Abdennahder, F. Kordon (Eds.), Reliable Software Technologies - Ada-Europe 2007. XII, 247 pages. 2007.

Vol. 4486: M. Bernardo, J. Hillston (Eds.), Formal Methods for Performance Evaluation. VII, 469 pages. 2007.

Vol. 4470: Q. Wang, D. Pfahl, D.M. Raffo (Eds.), Software Process Dynamics and Agility. XI, 346 pages. 2007.

Vol. 4468: M.M. Bonsangue, E.B. Johnsen (Eds.), Formal Methods for Open Object-Based Distributed Systems. X, 317 pages. 2007.

Vol. 4467: A.L. Murphy, J. Vitek (Eds.), Coordination Models and Languages. X, 325 pages. 2007.

Vol. 4454: Y. Gurevich, B. Meyer (Eds.), Tests and Proofs. IX, 217 pages. 2007.

Vol. 4444: T. Reps, M. Sagiv, J. Bauer (Eds.), Program Analysis and Compilation, Theory and Practice. X, 361 pages. 2007.

Vol. 4440: B. Liblit, Cooperative Bug Isolation. XV, 101 pages. 2007.

Vol. 4408: R. Choren, A. Garcia, H. Giese, H.-f. Leung, C. Lucena, A. Romanovsky (Eds.), Software Engineering for Multi-Agent Systems V. XII, 233 pages. 2007.

Vol. 4406: W. De Meuter (Ed.), Advances in Smalltalk. VII, 157 pages. 2007.

Vol. 4405: L. Padgham, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VII. XII, 225 pages. 2007.

Vol. 4401: N. Guelfi, D. Buchs (Eds.), Rapid Integration of Software Engineering Techniques. IX, 177 pages. 2007.

Vol. 4385: K. Coninx, K. Luyten, K.A. Schneider (Eds.), Task Models and Diagrams for Users Interface Design. XI, 355 pages. 2007.

Vol. 4383: E. Bin, A. Ziv, S. Ur (Eds.), Hardware and Software, Verification and Testing. XII, 235 pages. 2007.

Vol. 4379: M. Südholt, C. Consel (Eds.), Object-Oriented Technology. VIII, 157 pages. 2007.

Vol. 4364: T. Kühne (Ed.), Models in Software Engineering. XI, 332 pages. 2007.

Vol. 4355: J. Julliand, O. Kouchnarenko (Eds.), B 2007: Formal Specification and Development in B. XIII, 293 pages. 2006.

Vol. 4354: M. Hanus (Ed.), Practical Aspects of Declarative Languages. X, 335 pages. 2006.

Vol. 4350: M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, All About Maude - A High-Performance Logical Framework. XXII, 797 pages. 2007.

Vol. 4348: S. Tucker Taft, R.A. Duff, R.L. Brukardt, E. Plödereder, P. Leroy, Ada 2005 Reference Manual. XXII, 765 pages. 2006.

Vol. 4346: L. Brim, B.R. Haverkort, M. Leucker, J. van de Pol (Eds.), Formal Methods: Applications and Technology. X, 363 pages. 2007.

Vol. 4344: V. Gruhn, F. Oquendo (Eds.), Software Architecture. X, 245 pages. 2006.

Vol. 4340: R. Prodan, T. Fahringer, Grid Computing. XXIII, 317 pages. 2007.

Vol. 4336: V.R. Basili, H.D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, R.W. Selby (Eds.), Empirical Software Engineering Issues. XVII, 193 pages. 2007.

Vol. 4326: S. Göbel, R. Malkewitz, I. Iurgel (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. X, 384 pages. 2006.

Vol. 4323: G. Doherty, A. Blandford (Eds.), Interactive Systems. XI, 269 pages. 2007.

Vol. 4322: F. Kordon, J. Sztipanovits (Eds.), Reliable Systems on Unreliable Networked Platforms. XIV, 317 pages. 2007.

Vol. 4309: P. Inverardi, M. Jazayeri (Eds.), Software Engineering Education in the Modern Age. VIII, 207 pages. 2006.

Vol. 4294: A. Dan, W. Lamersdorf (Eds.), Service-Oriented Computing – ICSOC 2006. XIX, 653 pages. 2006.

Vol. 4290: M. van Steen, M. Henning (Eds.), Middleware 2006. XIII, 425 pages. 2006.

Vol. 4279: N. Kobayashi (Ed.), Programming Languages and Systems. XI, 423 pages. 2006.

Vol. 4262: K. Havelund, M. Núñez, G. Roşu, B. Wolff (Eds.), Formal Approaches to Software Testing and Runtime Verification. VIII, 255 pages. 2006.

Vol. 4260: Z. Liu, J. He (Eds.), Formal Methods and Software Engineering. XII, 778 pages. 2006.

¥371.-元

Table of Contents

Aspect-Oriented Requirements

Aspect Requirements to Design

Aspect-Oriented Architecture Design

Aspect-Oriented Domain Engineering

Panel

A Taxonomy of Asymmetric Requirements Aspects

Nan Niu[1], Steve Easterbrook[1], and Yijun Yu[2]

[1] Department of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 3G4
{nn,sme}@cs.toronto.edu
[2] Computing Department, The Open University
Walton Hall, Milton Keynes, UK MK7 6AA
y.yu@open.ac.uk

Abstract. The early aspects community has received increasing attention among researchers and practitioners, and has grown a set of meaningful terminology and concepts in recent years, including the notion of *requirements aspects*. Aspects at the requirements level present stakeholder concerns that crosscut the problem domain, with the potential for a broad impact on questions of scoping, prioritization, and architectural design. Although many existing requirements engineering approaches advocate and advertise an integral support of early aspects analysis, one challenge is that the notion of a requirements aspect is not yet well established to efficaciously serve the community. Instead of defining the term once and for all in a normally arduous and unproductive conceptual unification stage, we present a preliminary taxonomy based on the literature survey to show the different features of an asymmetric requirements aspect. Existing approaches that handle requirements aspects are compared and classified according to the proposed taxonomy. In addition, we study crosscutting security requirements to exemplify the taxonomy's use, substantiate its value, and explore its future directions.

1 Introduction

The many early aspects researchers and practitioners have grown a set of meaningful terminology and concepts, of which the most prominent is the distinction made between code-level and early aspects. Early aspects are concerns that crosscut an artifact's dominant decomposition, or base modules derived from the dominant separation-of-concerns criterion, in the early stages of the software life cycle [5]. "Early" signifies occurring before implementation in any development iteration, and embodies the key activities of requirements engineering, domain analysis, and architecture design, as indicated in the early aspects Web portal [12].

It is probably not a coincidence that one of the earliest descriptions of early aspects appeared in the proceedings of the premier requirements engineering (RE) conference in 1999 [14]. And the fact that the most recent RE conference

A. Moreira and J. Grundy (Eds.): Early Aspects 2007 Workshop, LNCS 4765, pp. 1–18, 2007.

(RE'06) presented the best paper award to an early aspects paper [17] demonstrates the strong connection between RE and early aspects communities. In fact, many existing RE approaches advocate and advertise an integral support of early aspects analysis, e.g., use cases [20], scenarios [3], viewpoints [28], and goals [36].

Aspects at the requirements level present stakeholder concerns that crosscut the problem domain, with the potential for a broad impact on questions of scoping, prioritization, and architectural design. A thorough analysis of early aspects in requirements offers a number of benefits:

- Explicit reasoning about interdependencies between stakeholder goals and constraints;
- Improving the modularity of the requirements structure;
- Identification of the impact of early aspects on design decisions can improve the quality of the overall architectural design and implementation;
- Conflicting concerns can be detected early and trade-offs can be resolved more economically;
- Test cases can be derived from early aspects to enhance stakeholder satisfaction; and
- Tracing stakeholder interests throughout software life cycle becomes easier since crosscutting concerns are captured early on.

Despite the growing awareness of these benefits and the continuing endeavor to achieve them, one challenge RE and early aspects communities currently face is that the notion of a requirements aspect, i.e., aspect at the requirements level, is not yet well established. We seek to explain and clarify the requirements aspects phenomena. The goal is to provide requirements analysts and other stakeholders a foundation for discussing specific challenges they might face in aspect-oriented RE projects. To this end, we conduct literature survey and domain analysis, thus presenting a taxonomy to show the different features of a requirements aspect. The sources of our survey focus mainly on asymmetric approaches and come primarily from the publications and reports in the literature, which can be found in [12].

Reference models with the unifying taxonomy represent prototypical models of some application domain, and have a time-honored status. One well-known example is the OSI 7-layer reference model, which divides network protocols into seven layers: physical, data link, network, transport, session, presentation, and application. The taxonomy of protocol layers is in widespread use, and is discussed in virtually every basic textbook on computer networks. The OSI 7-layer model is successful because it draws on what was already understood about the networks domain, thus codifying the core knowledge to be flexible.

Not every domain is sufficiently standardized to allow for a reference model or a unified taxonomy. In a blooming research field such as early aspects, people explore and tackle recognized problems complementarily, based on different mindsets and traditions. Various perspectives may not converge in a short period of time, which makes the conceptual unification process arduous and unproductive most of the time. As an example, the term "component" is used to describe

rather different concepts in software engineering: subsystems, JavaBeans, ActiveX controls, .NET assemblies, CORBA components, and more. As we shall see in section 2, the term "requirements aspect" saliently resembles in this respect the notion of a component.

Moreover, approaches like the one followed by the OSI 7-layer model are not necessarily supportive of change, especially when this change goes beyond the initially covered domain. However, at least experience on how to get to the model and the taxonomy should be recorded and shared [29], as undeniably a contribution to knowledge. Along this line, we present an initial attempt to identify reusable resources in the domain of requirements aspects. In section 3, We document our domain analysis results – a taxonomy of requirements aspects – using feature diagrams [21], and then apply the taxonomy to compare existing approaches that deal with aspects at the requirements level.

To exemplify the taxonomy's use and substantiate its value, in section 4, we present a reified requirements aspect – security, one of the most mentioned crosscutting concerns in the current literature. We then review related work in section 5, and conclude the paper with a summary and some directions for future work in section 6.

2 A Teaser Description for Requirements Aspects

There exist several definitions and descriptions for the term "requirements aspect", one of them by the initiators of research in early aspects: Elisa Baniassad, Paul Clements, João Araújo, Ana Moreira, Awais Rashid, and Bedir Tekinerdoğan:

> *"A requirements aspect, then, is a concern that cuts across other requirement-level concerns or artifacts of the author's chosen organization. It is broadly scoped in that it's found in and has an (implicit or explicit) impact on more than one requirement artifact." [5]*

We will use it as a starting point for our further discussion. Let's consider some parts of this description in detail:

- "a concern": Calling a requirements aspect a concern develops strong ties with stakeholders of the intended software system. A requirements concern is a matter of relevance that conveys the problem domain's property of interest to specific stakeholders. Therefore, any requirements aspect has its intent or purpose of existence, and needs to be traced to some stakeholder interest. Addressing requirements concerns thus enhances the stakeholder satisfaction and the overall software quality.
- "cuts across": No matter how stakeholder concerns are structured, a requirements aspect crosscuts the dominant decomposition, i.e., the author's chosen organization. This view assumes that the author has chosen some dominant organizing decomposition structure at the requirements level first, and makes the crosscutting concern a second-class object. Actually, aspect has become an equivalent substitute for a crosscutting concern in the literature.

- "author's chosen organization": Current requirements techniques offer a variety of structures for organizing the requirements, such as (structured) natural languages, use cases, scenarios, viewpoints, goal models, features, etc. [26]. The requirements analyst (author) develops a relatively well-organized set of requirements based on some dominant decomposition criteria and chosen structures. Requirements aspects emerge as a result of the lack of additional decomposition dimensions of the author's chosen organization.
- "broadly scoped": The authors of the original article [5] stated that broadly scoped properties can be quality attributes (nonfunctional requirements) as well as functional concerns that the requirements engineer must describe with relation to other concerns. These broadly scoped properties manifest as scattered and tangled concerns in the author's chosen organization of requirements.
- "impact on": Requirements aspects do not exist in isolation. They need to contact other requirement-level concerns or artifacts to provide some service according to their purpose of existence. This service providing process, which is also known as aspect weaving, is *usually* done in an oblivious fashion, i.e., the service provider (aspect) is impelled toward the consumers without them being aware of aspect's existence. Recent work has pointed out that obliviousness is neither an essential nor a desirable property of aspects [32]. Nevertheless, aspects need to explicitly specify the conditions, locations, and implications of the intended interactions.

This description of the term "requirements aspect" is very generic and thus it is not surprising that the term is used to describe rather different concepts: a collaboration in requirements for software components [14], an extension in a use case diagram [20], a softgoal in a goal model [36], an instance of terminological interference in viewpoints-based requirements models [25], a non-functional requirement (NFR) in a software requirements specification [17], and more.

The purpose of this paper is to try to clarify these different views by providing a taxonomy for requirements aspects. We therefore do not try to give one concise, closed definition. Instead, we will show the different features and characteristics a requirements aspect must or can have, thereby classifying the different kinds of requirements aspects as they are used today.

3 A Feature Diagram for Requirements Aspects

This section presents the results of applying domain analysis to existing approaches that tackle requirements aspects. Domain analysis is concerned with analyzing and modeling the variabilities and commonalities of systems or concepts in a given domain, thereby developing and maintaining an information infrastructure to support reuse [11].

We document our domain analysis results – a taxonomy of requirements aspects – using feature diagrams [21]. Essentially, a feature diagram is a hierarchy

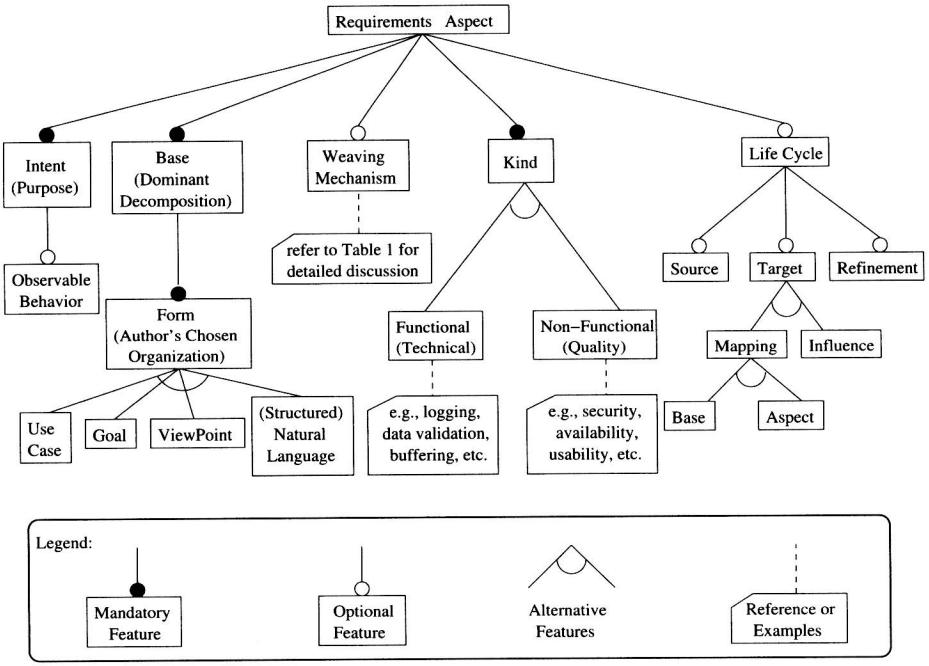

Fig. 1. A feature diagram for asymmetric requirements aspects

of common and variable features characterizing the set of instances of a concept [9]. In our case, the features provide a taxonomy and representation of design choices for approaches dealing with aspects at the requirements level.

We do not aim for this taxonomy to be normative and immune from change. In fact, the relatively new area of early aspects has already experienced many overloaded terms, and many of the terms we use in our taxonomy are often used differently in descriptions of other approaches. Consequently, we provide minute examinations of the terms and concepts as we use them. Furthermore, we expect the taxonomy to evolve as our understanding of requirements aspects matures. Our main goal is to show the vast range of available choices as represented by the current approaches, from a reuse perspective.

Figure 1 depicts a feature diagram we use as a basis for our discussion on requirements aspects. We deliberately restrict the diagrammatic notations in Fig. 1 to conforming with those originally proposed in [21], with reference and annotated examples provided to illustrate specific concepts. In particular, only three types of lines (edges) connecting boxes (nodes) are presented: mandatory, optional, and alternative (XOR-choice). This is because the notations in [21] are free of ambiguities, whereas later modified feature diagram variants are neither precise nor ambiguity-free [33]. While we will clarify the diagram semantics on the fly, we refer the interested reader to [21,9,33] for detailed discussion on feature modeling.

In Fig. 1, the features (denoted by the boxes) of the concept *requirements aspect* are described, which is located at the top of the feature diagram. The boxes directly connected to *requirements aspect* are the direct sub-features of a requirements aspect. The little circles at the edges connecting the features define the semantics of the edge. A filled circle means mandatory. Thus, every requirement aspect has an *intent* or *purpose* of existence, is of a certain *kind*, and cuts across multiple *base* modules according to the *dominant decomposition*. Optionally (denoted by the outlined circle at the edge), a requirements aspect provides *life cycle* information, and defines some *weaving mechanism*. Alternative features means an exclusive-or choice. For example, analysts organize requirements using one and only one base form in a particular RE stage like use cases, goals, viewpoints, and so forth.

We elaborate sub-features presented in Fig. 1 in the following sub-sections, and apply the taxonomy to compare different requirements aspects approaches. As long as we organize the discussion by (sub-)features, "aspects" emerge and crosscut this dominant decomposition structure. We therefore discuss these crosscutting properties in a separate sub-section.

3.1 Intent and Life Cycle

An aspect represents some concern, which only matters to specific stakeholders of a software system. A requirements aspect represents stakeholder interest in the problem domain, including high-level concerns like user satisfaction and happiness, system qualities like security and efficiency, software capabilities like fault tolerance and persistent storage, overall considerations like development time and return on investment, and many others. Each requirements aspect, therefore, needs to have an intent to help justify its very existence with particular stakeholder interest.

In many cases, the intent is further formulated and elaborated through observable behaviors to reflect what the stakeholder has in mind to expect the software to do or bring about. For example, to ensure security, the software system shall disable user accounts after the wrong password is entered three times. These observable behaviors are what make the intent operationalizable and measurable. Naturally, test cases can be derived to check whether the resulting software meets the intent, addresses the stakeholder concern, and guarantees the desired behavior.

Test cases present an instance of the *life cycle* feature, which supports traceability of requirements aspects throughout the software development life cycle. Requirements traceability is defined by Gotel and Finkelstein [13] as the ability to describe and follow the life of a requirement in both a forwards and backwards direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases.

Tracing a requirements aspect backwards to its origins helps uncover its intent and the source of stakeholder interest: whether the aspect is a concern