Gesellschaft für Informatik (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into the fields of
• Seminar
• Proceedings
• Dissertations
• Thematics
current topics are dealt with from the fields of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure the high level of the contributions.

The volumes are published in German or English.

Information: http://www.gi-ev.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-245-1

Enterprise software is one of the main research and innovation areas in the German State of Baden-Württemberg. The importance of enterprise software is expected to further grow in the future, since interrelations and interdependencies within and in between enterprises will increase. In order to design efficient business processes, the goal of the research initiative PRIMIUM was to systematically analyze the design and the usage of enterprise software solutions and make it usable for practice. This book presents some of the results of the research initiative PRIMIUM.

A. Heinzl, P. Dadam, S. Kirn, P. Lockemann (Eds.): PRIMIUM

151

GI-Edition

Lecture Notes in Informatics

Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)

PRIMIUM
Process Innovation for Enterprise Software

Proceedings

Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)

PRIMIUM
Process Innovation for Enterprise Software

15.04.2009
in Mannheim, Germany

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-151

ISBN 978-3-88579-245-1
ISSN 1617-5468

Volume Editors
Prof. Dr. Armin Heinzl
Lehrstuhl für ABWL und Wirtschaftsinformatik
Universität Mannheim
68131 Mannheim, Germany
Email: heinzl@uni-mannheim.de
Prof. Dr. Peter Dadam
Institut für Datenbanken und Informationssysteme
Universität Ulm
89069 Ulm, Germany
Email: peter.dadam@uni-ulm.de
Prof. Dr. rer. nat. Stefan Kirn
Lehrstuhl Wirtschaftsinformatik II
Universität Hohenheim
70599 Stuttgart, Germany
Email: wi2office@uni-hohenheim.de
Prof. Dr.-Ing. Dr.h.c. Peter C. Lockemann
Institut für Programmstrukturen und Datenorganisation (IPD)
Universität Karlsruhe
76128 Karlsruhe, Germany
Email: lockeman@ira.uka.de

printed by Köllen Druck+Verlag GmbH, Bonn

Preface

The federal state of Baden-Württemberg, Germany's high tech region, has developed a highly site-specific knowledge regarding Enterprise Software products and services over the past decades. Many people are familiar with SAP, the market leader for integrated business suites. In fact, there are almost 5,000 small and medium sized software and IT service enterprises in the state, indicating the vital role this business sector plays in the national economy. On the national level, 3% of the workforce in the software and IT service sector account for approximately 15% of the federal GDP. The total sales volumes exceeded 12 billion € in 2007.

A roundtable between industry leaders, politicians, and academics initiated by the state government of Baden-Württemberg and the industrial network "Baden-Württemberg Connected e.V." (bwcon) brought up the vital question how small and medium sized enterprise software vendors would be able to cope with changing market forces such as globalization and concentration. One venue of answers seemed to be quite simple: further improve the software products but also improve the software development process.

Improving the software product is related to better supporting highly specific business processes of the client organizations in order to create and to increase the added value resulting from enterprise software investments. Improving the software development process means to overcome the traditional, monolithic boundaries of software development within a company through inter-organizational collaboration in order to improve the quality and the time-to-market of the software product. Thus, the main idea was to transform the development process into a layered software development ecosystem which allows for the faster and better development and deployment of enterprise software.

Since industry leaders regarded this approach as a key opportunity, the state foundation of Baden-Württemberg which reinvests the privatization earnings for science and education projects, asked for bids of research-industry-consortia to address this issue. 16 consortia submitted a proposal of which three were selected by an independent industry and science jury, whose members came from other regions. The three consortia, which interacted with their industry partners as well as with the other consortia, formed the research network PRIMIUM (Process Innovation for Enterprise Software)[1].

This LNI edition compiles the key outcomes of the four years of work within this research network. It is organized along three sections. The first section deals with the better and faster specification of enterprise software. Only if the clients are better integrated into the requirements engineering phase, more specific and better software is likely to result. Thus, open proposal techniques, the alignment of software specifications with business objectives as well as privacy definition elements will be included. The second section focuses on the software development process itself. Contemporary architecture principles, integrated ontologies, automated workflows, traceability in model-driven architectures and current testing practices are the major elements of this part of the book. The third and final section highlights interdisciplinary elements of the software development process. Collaborative requirements engineering, end-to-end traceability and

[1] The acronym stands for "PRozessInnovation MIt UnternehMenssoftware", the German equivalent for "Process Innovation for Enterprise Software".

rationale management, and partnership networks in the software industry are topics presented in this volume.

The contributions presented in this monograph have been carefully selected and edited in a two-step process. First, a call for papers was issued for a workshop at the German Multi-Conference of Business Informatics 2008 in Munich. An independent jury reviewed and selected the best papers for presentations and guided the feedback process. Second, the final editing of the chosen papers and the structuring of this monograph was conducted by the editors of this book.

We would like to thank all the authors and members of the research network PRIMIUM for their contributions and cooperation. It has been a pleasure to collaborate with all of them! Our deepest gratitude belongs to the State Ministry of Science and Arts, Stuttgart, and Baden-Württemberg Connected which sponsored and supported this 3.5 million € project effort. Our thanks go to Dr. Heribert Knorr, Walter Kaag, Peter Castellaz, and Patrizia Illisson from the Ministry who stimulated the project and supported us in an exemplary way. Particular thanks also go to Klaus Haasis, Eike Bieber, and Tina Schanzenbach at bwcon who provided an excellent infrastructure for cooperation, communication and networking. A crucial role played the advisory board which selected the consortia as well as steered the work progress during the project. Our gratitude belongs to our colleagues Hans-Juergen Appelrath (Oldenburg), Joerg Becker (Muenster), Manfred Broy (TU Munich), Bernd Scholz-Reiter (Bremen), Elmar Sinz (Bamberg) and Robert Winter (St. Gallen) as well as Manfred Roux (formerly IBM), Harald Huber (USU), Martin Hubschneider (CAS) and Christian Sauter (Excelsis) who critically examined our research at each milestone and who provided thoughtful directions. Our special thanks also go to Thomas Kude and Lars Klimpke (Mannheim) who supported us diligently in editing this book. While we are very grateful to the authors of the chapters of the book, we take responsibility for the content and any errors. We hope this edition is an instructive and valuable primer for an important topic in software development.

Prof. Armin Heinzl	Prof. Peter Dadam	Prof. Stefan Kirn	Prof. Peter Lockemann
Mannheim	Ulm	Hohenheim	Karlsruhe

Contents

Bringing Developers and Users closer together: The OpenProposal story

Asarnusch Rashid, Jan Wiesenberger, David Meder, Jan Baumann

FZI Forschungszentrum Informatik
Research Center for Information Technologies at the University of Karlsruhe
Haid-und-Neu Str. 10-14
D-76131 Karlsruhe
rashid@fzi.de, wiesenberger@fzi.de, meder@fzi.de, baumann@fzi.de

Abstract: Even though end-user participation in requirements engineering (RE) is highly important, it is at present not frequently used. Reasons can be found in the large expenditure of time for organizing and carrying out surveys as well as in the time it takes to understand the users' requirements. This research is supposed to address this problem by presenting the OpenProposal approach for distributed user participation using visual requirement specifications. First experiences made in several case studies show the potential and limits of this approach and outline the possibilities of application.

1 Introduction

It is well known that Requirements Engineering (RE) is one of the biggest challenges, and that all stages of RE dealing with elicitation, specification, prioritization and management of requirements are vitally important for the success of a software project. As RE is a complex process involving many people, a lot of different methods and tools were developed to support this highly collaborative process. In the research project CollaBaWü[1] the researchers were confronted with the task to evaluate existing methods and tools for practical usefulness in cooperation with software companies and financial industry and if necessary to develop new methods and tools. In this context, the issue of user involvement in RE has become apparent in many conversations and analyses with the industrial partners.

The aim of user involvement in RE is to improve the design process, facilitate the implementation and to address ethical principles [NJ97]. User involvement can be performed when developing requirement specifications, validating requirement specifications, supporting detailed design and development, reviewing specifications, inspecting prototypes and accepting released products. A literature review [KKL05] shows that user involvement is found to have a generally positive effect on system

[1] www.collabawue.de: 'CollaBaWü' is a project (2004 – 2007) commissioned by Landesstiftung Baden-Wuerttemberg foundation, which aims at increasing the overall productivity in the development lifecycle of enterprise applications. In this respect, the main objective of the project is to promote industrialisation in enterprise application development focussing on the particularities of the financial service provider domain.

success and on user satisfaction, and some evidence can be found suggesting that taking users as a primary information source is an effective method of requirements elicitation [CM96, EQM96, KC95, Ku03]. In addition, involving users in RE could have an important role, since users are nowadays recognized as being one of the essential groups of stakeholders, while a lack of user involvement was shown to lead to serious problems in RE [Po95]. These results coincide with the experiences of industrial partners (each with more than 10.000 users) in the project CollaBaWü. Their software department aim at a closer relationship with their users in order to improve their RE processes.

Methods already in practical use include User Experience, Usability Workshops, User Support Units and Employee Suggestion Systems. But recognizing that these procedures are too formal and heavy-weight, and the insight that systematic approaches to understand users' needs and continuous user involvement are still lacking, lead to the suggestion that users have to be able to participate with small effort – the optimum being during their daily work - and developers have to be able to obtain enough valuable information without additional effort. By collaborating, users should exchange and discuss their ideas in a shared environment. Further, transparency is desired, meaning that users can track the development of the suggestions they submitted.

This research aims to contribute to the existing approaches by presenting the OpenProposal system for distributed user participation. The fundamental idea behind OpenProposal is based on the fact that in most modern software products the users' requirements refer directly to the graphical user interface. Therefore the idea of capturing these requirements in a graphical form, supplementing a textual description, was taken into consideration. Since nowadays graphical annotations are unusual in requirements management, and since most of the time sketches and screen shots are merely used on paper, this research wants to discuss the use of graphical annotations in Requirements Management (RM).

For this it is aimed

- to understand the users' and the developers' needs in requirements management,
- to develop a new methodology and a new concept of IT support to enhance existing practices,
- to develop a certain formal notation language providing a language to enable users and software developers to formulate and discuss users' requirements, and finally
- to evaluate the methodology and IT support concept in order to identify chances, risks and limits.

The paper is structured as follows. At first, the theoretical background and related Information System Development (ISD) approaches are discussed. Secondly, the process, concept and architecture of OpenProposal are presented. Then experiences gained from case studies are outlined. Fourth, possible fields of application of

OpenProposal are described and finally, the lessons that could be learned from this research are discussed.

2 Related Work

The OpenProposal concept was able to take its inspiration from several different fields of research, including Participatory Design (PD), Requirements Engineering (RE) and Digital Annotation (DA). The idea for user participation came from the field of PD, as the methods for this particular type of end user integration primarily originate from it [Su93]. PD is solely concerned with user participation in all phases of system development. In general, existing methods for user participation during requirements elicitation utilize direct face to face communication or prefer user interviews. Even though we do not question that these methods allow a complete requirements specification, the successful execution requires considerable effort, both from the side of the developers as well as from the side of the users [DESG00]. Furthermore the increasing degree of interconnectedness through the Internet provides new possibilities of cooperation, but they also imply the need of addressing the challenges of globalization in software development. During the recent years, the employment of CSCW software (Computer Supported Cooperative Work) has therefore been discussed more intensely in PD [KB03]. New approaches in user participation like the concept of Participation Design in Use [SD06] or the new area of Distributed Participatory Design (DPD) [DND06] are supposed to allow the usage of methods of PD in distributed (over time and space) working environments.

In RE research there are a lot of methods and tools available, which support the process of requirements acquisition [HL01]. Besides the traditional concepts of user integration mentioned above, there are many approaches to alternative specification techniques in order to achieve end-user participation. Reeves and Shipman [RS92] describe a combination of visual and textual elements. Li et al. [LDP05] have developed a solution, which uses Natural Language Processing. Video techniques are used in the scenario "Software-Cinema" of Bruegge et al. [Br04], which addresses requirements acquisition in mobile environments. There are also numerous commercial tools supporting visual specification of requirements. The best known notation method is UML, which promotes the use of Use-Case diagrams during the RE phase of software development. These formal techniques support primarily software engineers and presume skills in modelling languages.

Other visual aids in RE include mock-ups [Ma96] and Rapid prototyping techniques [BH98], which are commonly used in early phases of software projects. They do allow software engineers to sketch screens with low functionality as well as to run first usability tests. Tools supporting these techniques are applied by engineers and analysts but not by the user himself. They are merely enabled to return feedback in the usual way of face-to-face meetings. An approach centring more on the user is offered by Moore [Mo03]. It is based on requirements acquisition using GUI elements without functionality. End-users 'will create mock user interface constructions augmented with textual argumentation that will act as communication to software requirements

engineers' [Mo03, p. 2]. This approach allows users to construct their own user interfaces but still it has to deal with the problem that real software is very complex and end-users will not have enough time to construct a whole software system.

The advent of graphical user interfaces has led to requirements concerning the modification and improvement of such an interface. Tools supporting this process often use DA. The tool Annotate Pro [An06], for example, provides several functions to draw annotations directly on the users' screen by using snapshots in combination with ordinary picture editing functionality. This enables end-users to draw comments on their active applications, without time-consuming trainings and preparations. The commented snapshots serve as requirement specification and can easily be sent to the software engineers by email. As this tool neither contains a method which follows a well-structured plan nor provides a formal notation language there does not exist any common language either which means that users are free to paint sketches and to send them to anyone without assistance. It is assumed that developers may have difficulties in understanding these paintings. Furthermore, there is no possibility for end-users to track the submitted requirements. There are other tools of a similar nature, but addressing different aspects. JING [Ji07], for example, focuses on fast sharing of annotated images and videos. It uses a very basic set of tools, which are nonetheless sufficient for marking and explaining what the central aspects of this picture are. While offering no annotation possibilities for videos, providing a video capture possibility makes the tool flexible. JING also features built in support for upload to Flickr, ScreenCast or any user definable FTP server. However it shares the same shortcomings that Annotate Pro has. Similarly the usability testing suite Morae [Mo07] does not support the structure of the test conducted, nor does it directly support tracking of submitted requirements. It focuses on recording and analyzing usability studies and features a large set of recording options, logging and observation functions as well as tools for analyzing results and creating reports.

The literature about DA states that annotations are not only useful because they allow capturing the application concerned, but they do also support such crucial mental functions such as remembering, clarifying and sharing, as Bottoni et al [BLR03] point out. Their paper provides a formal analysis of digital annotation and identifies operations and procedures that an annotation system should allow. Annotations are also important for a two-way information exchange, as discussed by Fogli et al [FFM04], who also define the tools required to support creation and the use of annotations.

In summary, the problem is that none of the present tools and approaches provides all essential functionality to support end-users in an adequate way. All of them are lacking either usability or efficiency or collaboration. This research is supposed to address this problem by presenting the OpenProposal approach for distributed user participation using visual requirement specifications.

3 OpenProposal: Process, Concept & Implementation

OpenProposal aims at aiding users to express their ideas about how an application might be enhanced. At the same time it is also supposed to help developers by imposing a structure on the annotation process which will make it easier for them to grasp the users' intention. In order to achieve these requirements, the OpenProposal tool differs from other annotation tools, e.g. Annotate Pro [An06] in so far as the users do not interact with a set of free-form drawing tools, but with a toolset representing possible changes he wants his target application to undergo.

OpenProposal is supposed to allow users to annotate their feature requests, error reports or enhancement requests directly on their applications workspace and send these requests to the requirements management. Lots of communication problems can thus be avoided – e.g. misconceptions due to wrong choice of wording, incomplete data, descriptions which are too elaborate – which often arise from text-only communication like E-mail or the internal employee suggestion systems. The aim of OpenProposal is to integrate users efficiently into the development process during their daily routine when using the application, to reduce the usual effort associated with participative requirements elicitation and to allow a high degree of implementation of the captured requirements with the help of structured recording. Furthermore, OpenProposal is supposed to increase the transparency of the requirements management process, thus ensuring motivated participation of as many employees as possible during requirements elicitation.

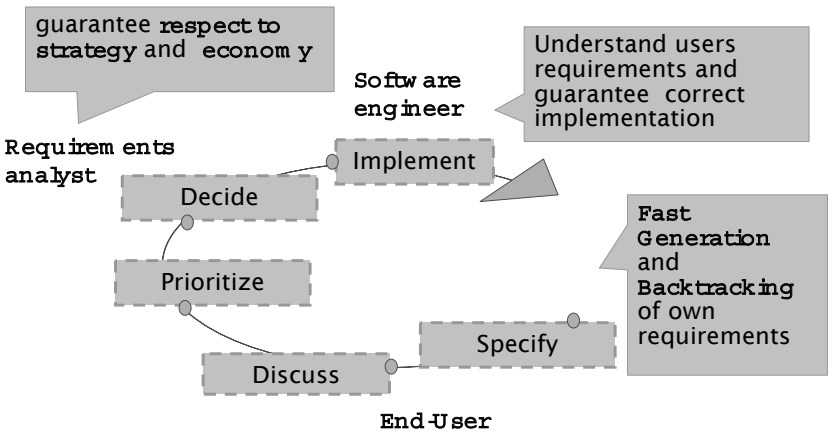

Figure 1: The OpenProposal Process

3.1 Process of OpenProposal

The OpenProposal process (see Figure 1) centres around five actions, specify, discuss, prioritize, decide and implement, and three roles, end user, requirements analyst and software engineer. Each role has its own special set of requirements and participates in a

certain subset of the five actions. The end user requires a possibility to generate his own requirements in a fast way. He also wishes to track progress on his requirements. End users take part in the specification and discussion of requirements. The requirements analyst needs to guarantee that whatever is done respects the company's overall strategy and is economically feasible. He takes part in the discussion of the proposals, prioritizes them and decides on which proposal will be implemented and which will not. He may also propose his own ideas and have them discussed with the other stakeholders. The software engineer has as a requirement the need to understand the users' proposals and to guarantee their correct implementation. He can submit his own proposal specifications, participate in discussions to contribute with his professional knowledge of what is technically possible and feasible, and is responsible for implementing the proposals that have been decided on.

3.2 Concept of OpenProposal

To ensure the involvement of users in distributed RE being successful, a system is needed permitting users to formulate their proposals with simple tools, to submit the created proposal to the developers and to track the proposal's progress. Furthermore the decision maker and the software developer should be able to manage and edit the proposals in an efficient way in order to benefit from the possibilities of the RE system, too.

OpenProposal is a software system which is supposed to fulfil these criteria. It should be possible to use it in conjunction with any established requirements analysis procedure currently employed in the company, which will thus be extended with efficient user involvement. Users should be enabled to create and discuss proposals for existing software as well as software currently being under development and it should be possible to propose improvements as well as new features. The level of detail is up to the user.

The OpenProposal process provides two tools. The annotation tool enables the user to visually formulate his ideas and send them to the collaboration platform tool, which gives an overview of the submitted proposals and allows discussions between users, developers, deciders etc. The process can be initiated in two ways. One way is to explicitly call for user participation, mostly for software which is currently under development. The other way is that the user wants to submit a proposal for an application without external motivation merely wishing to improve the software.

One essential benefit for the user is that he can actively participate in the process of software improvement and is thus able to shape the application the way he wants to. The user employs the annotation tool for the fast generation of graphical requirements and submitting them to the collaboration platform. The user can then track the progress using the collaboration platform.

The system is used by the decision maker to collect and consolidate the users' requirements and to compare them with the strategic and economic targets of the company. If the requirements are collected globally, covering all company divisions, he can detect and use synergies between the divisions. Using the collaboration platform, he

can receive an overview of the requirements, can discuss them with users and developers and thus determine the priorities.

The developer benefits from being able to participate in the discussion at an early stage, to inform users, decision makers and analysts about the technical possibilities and restrictions. The graphical specification is supposed to improve the process of understanding what users want, and implement the proposals in the correct way.

3.3 Architecture of OpenProposal

The OpenProposal implementation consists of an annotation tool, an XML specification, an Issue-Tracker and a mediator. As can be seen in Figure 2, the annotation tool gathers annotated screenshots which are stored together with the individual annotation objects in an XML specification. This specification also contains metadata about the proposal as well as the user's system. This specification is sent to a mediator specific to the Issue-Tracker. The mediator takes the information from the specification which will be directly entered into the Issue-Tracker software and creates a new issue with it, attaching the screenshot image and the XML file in the process. Stakeholders can log into the Issue-Tracker to rate and discuss proposals. This information can then be requested by the annotation tool through the mediator from the Issue-Tracker, in order to present users of the annotation tool a list of previous annotations of that application with their ratings and discussions.

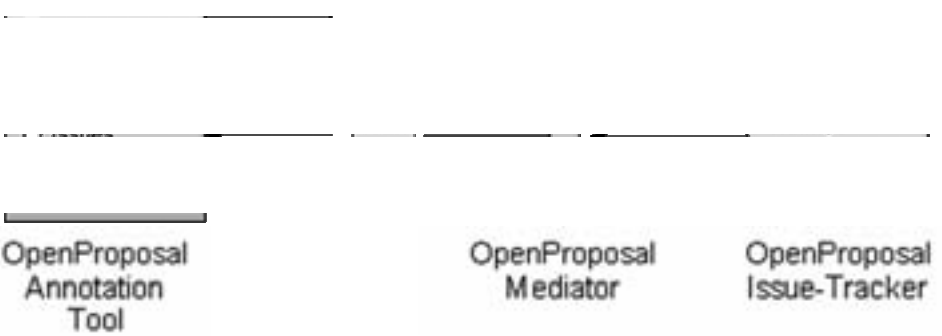

Figure 2: The OpenProposal Architecture

3.3.1 User Interaction: Annotating and Handing-In of Issues

Figure 3 exemplifies the functionality of OpenProposal and illustrates the way of annotating with OpenProposal. Imagine a user writing a document with Microsoft Word. The user has some ideas for improvement and starts OpenProposal. First, the OpenProposal notepad is opened and the user is asked to choose a category for his suggestion (C) in section A (I). Then, section B (II) is selected. The screen of the application is captured automatically and the user can sketch his suggestions for improvement directly on the screenshot. The toolbar offers four annotation related tools. The "Add" tool (1) allows users to specify a position where they would like to have a new object on the screen. The "Remove" tool (3) is the inverse; an existing element is marked as superfluous. With the "Move" tool (2) users first select an area which should be moved to a different place in the applications workspace, then to the new target area. The "Comment" tool (4) can be used for suggestions the other tools are not able to express directly, as well as refining and adding further detail to the other tools' annotations. Users may pause the annotation, e.g. if they want to change the layout of the application they are annotating (A).

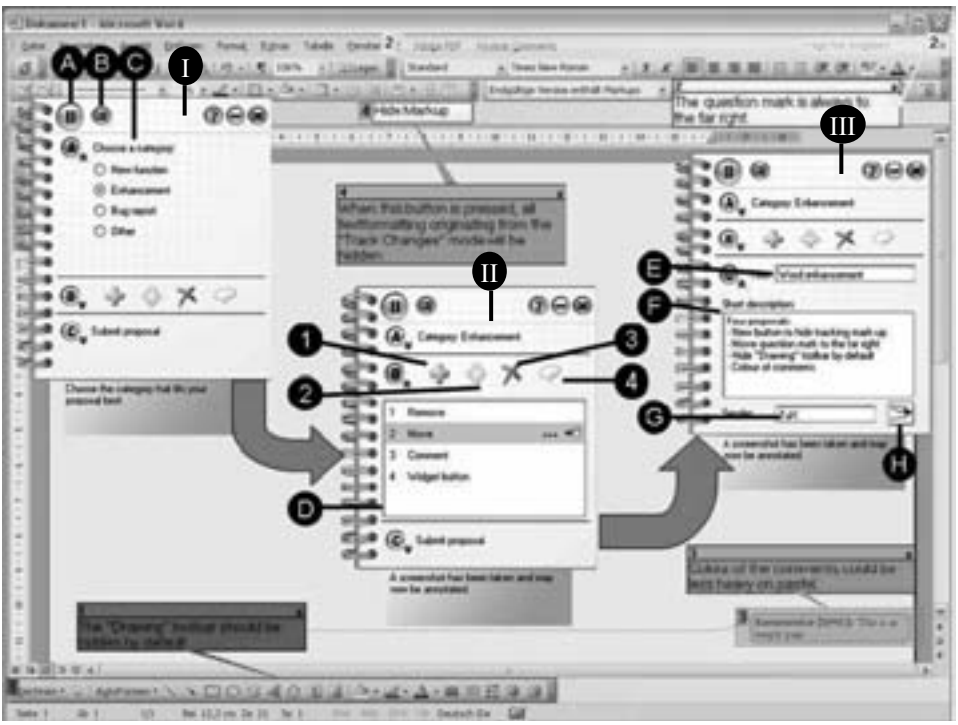

Figure 3: OpenProposal tool bar

All annotations are represented as objects which may be edited, moved or deleted whenever the users want to (D). Once they finish their annotations, users can send their requests to the issue-tracker (H). Prior to sending, the users are prompted to give their

request a title (E), a text description (F) and their usernames (G) in section C (III). Users may exit the application at any time. By pressing the Button Specification List (B) users can access to the window illustrated in Figure 5. The functionality of this window is described in the next chapter.

3.3.2 Collaboration: Viewing, Discussing and Rating Issues

The data provided by users is stored in the collaboration platforms' database, which may be accessed via a web front-end. When logging into the collaboration platform, the users are first presented with a list of all issues entered. When selecting one of the issues in the list, the issue window is shown (Figure 4). Here the users will be able to read a description of the issue (I), view details such as the status or the priority (L), participate in the discussion about this issue (K) and take a look at the annotated screenshot associated with this issue (M).

Figure 4: Web-based Issue Tracker

To support the user, the annotation tool of OpenProposal offers a window with a list of all issues (Figure 5) which have been created for the application. This makes it possible to view submitted issues directly in the annotation tool, without the need to open the web-based issue tracker. Because of the large number of specifications, the user is not able to view each specification in the list. In order to reduce the lists' size, OpenProposal provides a filtering function (N). This specification filter works as follows:

1. First all running applications are stored. If a running browser is detected (currently Internet Explorer, Firefox and Opera are supported), the application retrieves the address of the active website.

2. Then the topmost window on the screen is determined. If the user has already created some annotations, the filter also determines the applications which the user has annotated.

3. Lastly, the specifications belonging to one or more applications or website addresses which were determined in the two steps above are shown.

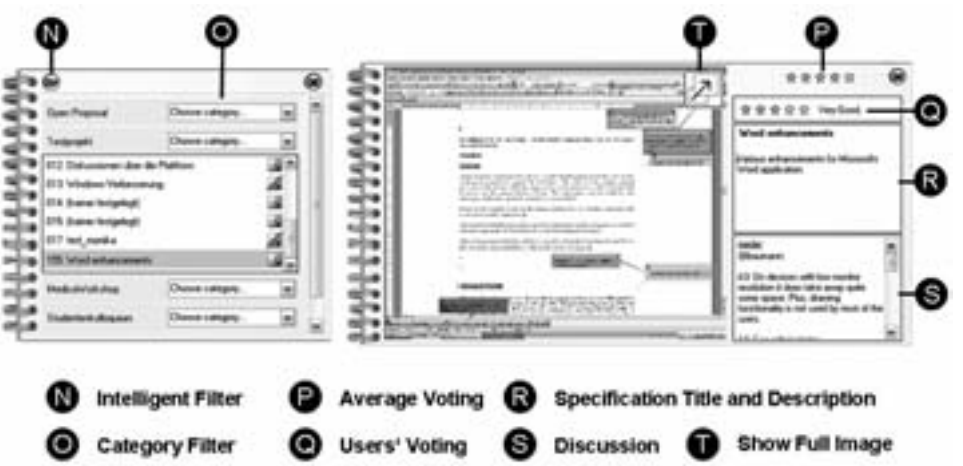

Figure 5: Specification list

The filtered list will be noticeable smaller than the unfiltered list and the user can see through the specifications more easily. Additionally the users can vote for each specification by giving a rating from one to five to express his conformance with the specification.

4 Experiences with OpenProposal

In the course of this research OpenProposal has been evaluated in several steps. First, usability tests were performed to ensure the usability of the annotation tool. Next, case studies with the software development department of the company TRUMPF and a department of the University of Karlsruhe were set up. At present, several case studies in cooperation with small companies specialising in usability design and software development as well as software development departments of larger companies have been started.

4.1 Usability Test of OpenProposal

In February 2007 a first usability and user test was performed with the goal of evaluating the current version regarding usability and user satisfaction. The results of the test would also be used as a basis to create the next version of our OpenProposal application. The 16 test subjects consisted of students from the University of Karlsruhe and employees of the FZI Forschungszentrum Informatik in Karlsruhe, Germany. The test was centred around five annotation tasks the subjects had to perform on an old version of the Firefox[2] browser. A short pre-test interview and a long post-test questionnaire where used to gather information about users thoughts and expectations of the system, as well as the degree to which these expectations where fulfilled. Test subjects where monitored by the investigator the whole time in addition to a video and screen capture being recorded.

The results of the interviews, the observations of the investigator, an analysis of the video recordings and the questionnaires yielded numerous proposals for enhancements, ranging from start-up behaviour (e.g. OpenProposal used to switch to annotation mode directly after program start, freezing the users screen in the process – this was later changed because of empirical evidence) to interface refinements (e.g. the previous version used a separate object list, user demand was an integrated list). But OpenProposal also received encouraging ratings concerning ease of use and usefulness of its annotation concept, for example when being asked if a tool for graphical creation of proposals for software should be provided (averaging to “agree” on a five point scale ranging from “strongly agree” to “strongly disagree”). All in all, the users rated the software as a whole with “good” on a five point scale ranging from “very good” to “very bad”. The enhancement proposals as well as the questionnaire results formed the basic set of changes to be implemented in OpenProposal 2.0.

New possibilities for further studies were found as well. For example, a question during the interview was “What advantages do you think OpenProposal would have?” to which test subjects replied, that the creation of proposals would be faster with this application. This assumption was tested against traditional methods, by taking the time both need for a given task. Some of these questions were addressed in the subsequent case studies, others required special setups or a long evaluation time and could not be answered yet.

4.2 Case Study ‘TRUMPF’

In September 2007 a second test was done at the TRUMPF Company. TRUMPF is a high-tech company focusing on production and medical technology. The TRUMPF Group is one of the world's leading companies in manufacturing technology, with sales of 1.65 billion/US$ 2 billion and approximately 6500 employees. Since efficient fabrication of high quality components is not a question of hardware alone, TRUMPF also develops the software systems for their hardware. Usability workshops are a part of the software development cycle at TRUMPF, and such a workshop was used to evaluate our new OpenProposal 2.0 which was used to create proposals for the software being tested during the workshop.

[2] http://www.mozilla-europe.org/de/products/firefox/

This short case study encompassed 11 test subjects in total and was carried out over two days. There were two groups on each day, one using OpenProposal, the other using another interface for proposals provided by the issue tracking software used by the TRUMPF software development department. The groups were set up so that every test subject would be able to use both interfaces for his proposal, and at the end of each day the subjects were given a questionnaire specific to the interface they used that day. Besides the analysis of the questionnaires' results, the proposals themselves were evaluated and the developers were interviewed. Additionally, an investigator monitored the study and wrote down comments, problems and observations he made during the workshop.

Question	OpenProposal	Tracker Interface
I was able to quickly find my way in …	Agree	Agree
I often got stuck using … and had to find a work around.	Strongly disagree	Disagree
… shows too much information at once. I found this confusing.	Strongly disagree	Disagree
Proposals are easy to create and don't require much mental effort	Agree	Agree
I made mistakes using …	Disagree	Disagree somewhat
Symbols and naming are easy to understand in …	Agree somewhat	Undecided / Disagree somewhat
The structure of the interface is easy to understand	Agree	Agree
Creating proposals was unnecessarily complex and took a long time.	Strongly Disagree	Disagree

Table 1: Results of the case study 'TRUMPF'

The results showed that OpenProposal was in general well received by the participants. Eight questions on the questionnaire were asked twice, once for OpenProposal and once for the tracker interface (see Table 1). The items were measured on a seven point Likert scale with the options "strongly disagree", "disagree", "disagree somewhat", "undecided", "agree somewhat", "agree", "strongly agree". The first item, "I was able to quickly find my way in …", had an average rating for both systems of "agree". The second item "I often got stuck in … and had to find a work around" received average ratings of "strongly disagree" for OpenProposal, which was thus a bit better than the average of "disagree" for the tracker interface. Similarly the third item "… shows too much information at once. I found this confusing." also received an average of "strongly disagree" for OpenProposal and "disagree" for the tracker. The fourth item, called "Proposals are easy to create and don't require much mental effort", was again rated

“agree” for both systems. Item five, “I made mistakes using … “, was rated with “disagree” for OpenProposal and with “disagree somewhat” for the tracker, again OpenProposal received a slightly higher rating. The biggest difference was found at item six “Symbols and naming are easy to understand in … “, where OpenProposal received an average rating of “agree somewhat” and the tracker was rated between “undecided” and “disagree somewhat”, another rating where OpenProposal comes out on top. Item seven, “The structure of the interface is easy to understand”, received an “agree” rating for both systems and at the last item, “Creating proposals was unnecessarily complex and took a long time.”, OpenProposal received another slightly better rating: “strongly disagree” as compared to “disagree”. The list of “must fix” enhancement requests was noticeably shorter as well. The participants noted that they did not see OpenProposal as a replacement of the existing tracker interface, but rather as an easy-to-use alternative frontend. The software was so well received, that by now it has become an inherent part of the usability process at TRUMPF.

4.3 Case Study ‘University of Karlsruhe’

In November 2007 a third usability and user test was launched at the IISM (Institute of Information Systems and Management) at the University of Karlsruhe. The software being tested was a new content management system which would be responsible for the institutes web pages and intranet services. The test phase of the system would be at least two weeks, and users were encouraged to transfer data from the old system to the new during that phase and report any problems or errors they found in the process. To ease the reporting process, OpenProposal would be provided to all users and would be configured so proposals are directly sent to the issue tracker included in the content management system.

In a first step, the first usability and user test was replicated with several test subjects from IISM. This was done to evaluate, whether the new version was indeed an improvement over the old version. The second step began with the introduction of the new content management system as well as OpenProposal in mid of November. A time frame of two weeks was given to the participants to get used to the new system, file bug reports, enhancement and feature requests and transfer data. This phase of the test is still in progress because the deadline of the introduction of the content management system had to be rescheduled to May 2008.

Test Item	Median Feb07	Median Nov07	p
Handling the user interface was [easy, medium, hard]	medium	easy	4,67 %
There are [none, some, many] functions I miss in OpenProposal	some	none	0,50 %
OpenProposal sufficiently informs me about what it is doing at the moment [strongly agree, agree, neutral, disagree, strongly disagree]	neutral	strongly agree	2,38 %

OpenProposal has a persistent style of handling throughout the whole program [strongly agree, agree, neutral, disagree, strongly disagree]	agree	strongly agree	3,71 %
Technical performance of OpenProposal was [very good, good, neutral, bad, very bad]	neutral	very good	1,52 %

Table 2: Results of the Case Study 'University of Karlsruhe'

Only results of the replication test are available as of now. Due to time constraints only five test subjects could be interviewed and observed. A Mann-Whitney-U test was performed for each question on the two sets of answers (one from February, one from November). This statistical non-parametric test can be used to check if two samples have equal distributions, in this case meaning that if the test yields a significant result, the sets of answers can be considered statistically different. According to Albers et al [AKK07] the level of significance is usually set to 5% for significant results and 1% for highly significant results. For five of the 30 items the Mann-Whitney-U test calculated a significant difference ($p<5\%$), all other differences in sets of answers were not significant (see Table 2). The first statistically different item was "Handling the user interface was … " with the three choices "easy", "medium" and "hard" being possible. While in February participants on average answered with "medium", in November the average answer was "easy", the difference being statistically different with a significance of p=4,67%. The second item was "There are … functions I miss in OpenProposal" with the three options "none", "some", "many" where the first test in February had an average result of "some" while the second test in November had an average result of "none", the difference being highly significant with a significance of p=0,50%. The third item was "OpenProposal sufficiently informs me about what it is doing at the moment" with the five options "strongly agree", "agree", "neutral", "disagree" and "strongly disagree". The average answer in February was "neutral", while in November the participants on average choose "strongly agree", the significance being p=2,38%. When being asked about persistency of handling, "OpenProposal has a persistent style of handling throughout the whole program" with the options "strongly agree", "agree", "neutral", "disagree", "strongly disagree", the participants of the first test in February answered on average with "agree" while in November the average answer was "strongly agree". The difference had a significance of p=3,71%. Lastly, the item "Technical performance of OpenProposal was …", with the choices ranging from "very good", "good", "neutral", "bad", "very bad", was rated with an average of "neutral" in the first test, while the November test had an average rating of "very good", the significance of this difference being p=1,52%. This shows that in all significantly different results, the new version received better ratings than the old version and can thus be considered an improvement.

5 Possible Fields of Application of OpenProposal

OpenProposal is supposed to support the RE phase of the software development process. During the course of the research project, it became apparent that the most promising

fields seemed to be ‘Usability Tests in Usability Workshops’, ‘Support and Maintenance of Software’ and ‘Global Software Development’.

5.1 Usability Test of Software in Usability Workshops

OpenProposal was built with a non-intrusive integration into the users’ workflow in mind. A field of application where this is especially helpful is that of usability workshops. The purpose of these workshops is an evaluation of an existing piece of software using people which correspond to the actual user as well as possible. The most important sources of information in these workshops are the test subjects themselves, especially their suggestions and ideas for improving the software at hand. Traditionally, when making such a suggestion the test subject would either make a handwritten note on a piece of paper, switch application context to write an electronic note or report their suggestion to the investigator. The best traditional way would be the third option, since it allows questions for clarification and refinement of the new proposal. This however would require having close to as many investigators as there are test subjects, making this option expensive. Having the test subject switch application context makes writing proposals cumbersome for the subject, since he may need to switch back and forth a number of times to write a good proposal. This can be solved when using handwritten notes. Here however the problem arises, that these notes need to be deciphered and converted into an electronic format. While OpenProposal cannot replace an investigator, when time and money are an issue it is likely to perform better than both alternative methods. The user does not have to switch context to some other application, but creates his annotation directly inside his current context. And there are no handwritten notes which need to be processed after the workshop improving clearness and correctness of the resulting proposals.

5.2 Support and Maintenance of Software

When considering software support, OpenProposal allows the end user to graphically formulate his problem – if it is referring to the user interface - and send it to support, for example via e-mail. The support team can then quickly detect the user’s problem without reading a long textual description of the problem. On the other hand OpenProposal can also be used by the support team, helping to translate the user’s support request into a graphical specification and sending it onto an Issue-Tracking platform. Similarly, OpenProposal can be used for software maintenance, where both end users as well as developers can file bug/error reports and improvement proposals as well as discuss the existing proposals.

5.3 Global Software Development

Modern software products often tend to be highly complex. Their development and production requires a lot of expertise and competence that can rarely be found in one place, at least not with the economy necessary for a fiercely competitive, global world. Thus, modern software production tends to be highly fragmented, geographically

distributed on a more or less global scale, where each participant is specialized in its own core competence. The resulting "global" software products should have the same quality at a more attractive price compared to those one would achieve at a single place. The design, the development and the production processes for a global software development involve new competencies in communication, collaboration, integration, and technical and managerial control.

There are some problems in global software development, particular to communication or information sharing. Problems like time zone difference or geographical distance hinder successful communication. Nowadays these problems are reduced by the wide availability of modern communication techniques like the internet.

OpenProposal supports the requirements engineering in geographically distributed environments. The tool supports the communication between the project team, users and customers occur in a geographically distributed way using modern communication techniques to reduce the impact of the geographical dispersion, e.g. different time zones. By generating requirements descriptions it supports the user in a formal and understandable way.

6 Summary & Outlook

In this research, the concept and first evaluation of OpenProposal are presented. All the conclusions drawn so far are based on software companies' practical knowledge as well as on previous related research and the results of the first case studies. Major findings of our research are the conception of the OpenProposal methodology for acquisition and management of user feedback in software development projects, the implementation of a tool support and the evaluation of the elaborated concept and implementation.

Our research has shown that the OpenProposal approach can help to improve the communication processes in software development projects. The usability tests revealed users' acceptance of the OpenProposal concept and the sufficiency of the functionality of the OpenProposal annotation tool for users' needs. In the case study 'TRUMPF' OpenProposal was successfully realized in a real life scenario and is still in use. In the view of Users, Designer and Developer of TRUMPF OpenProposal performed better than usual methods. It seems possible that this work can reveal new findings about the way users and software developers interact and can therefore offer new opportunities for innovative ways of collaboration in RE e.g. corresponding methods in Global Software Development.

The evaluation also point out open issues of the OpenProposal approach. In future studies we need to focus our research on the view of developers and deciders and improve the management and assessment of OpenProposal annotations.

The limits of our approach are clear: It was never our goal to argue that OpenProposal is the best and only solution. It can be only an additional methodology in users'

involvement and cannot replace interviews and usability workshops, but it can be a reasonable supplement.

References

[AKK07] Albers, S.; Klapper, D.; Konradt, U.; Walter, A.; Wolf, J.: Methodik der empirischen Forschung, Gabler Verlag, Wiesbaden, Germany, 2007.

[An06] Annotate Pro, http://www.annotatepro.com/, viewed on 13.06.2006.

[BH98] Beynon-Davies, P.; Holmes, S.: Integrating rapid application development and participatory design, In: IEEE Software, Volume 145, Issue 4, pp. 105-112, 1998.

[BLR03] Bottoni, P.; Levialdi, S.; Rizzo, P.: An Analysis and Case Study of Digital Annotation. In: Bianchi-Berthouze, N. (Eds.): Proc. 3rd International Workshop on Databases in Networked Information Systems, Aizu-Wakamatsu, Japan, 2003, pp. 216 - 230. Lecture Notes in Computer Science 2822, Springer, Heidelberg, Germany, 2003.

[Br04] Bruegge B.; Creighton, O.; Purvis, M.: Software Cinema, CHI Workshop on Identifying Gaps between HCI, Software Engineering and Design, and Boundary Objects to Bridge Them, Vienna, Austria, 2004.

[CM96] Chatzoglou, P.C.; Macaulay, L.: Requirements Capture and Analysis: A Survey of Current Practice, Requirements Engineering, Volume 1, Issue 2, pp. 75-87, 1996.

[DESG00] Damian, D.E. H.; Eberlein, A.; Shaw, M.L.G.; Gaines, B.R.: Using different communication media in requirements negotiation, IEEE Software Volume 17, Issue 3, May-June, pp. 28-36, 2000.

[DND06] Danielson, K., Naghsh, A.M., Dearden, A.: Distributed Participatory Design. Extended Abstract of the workshop for Distributed Participatory Design at conference NordiCHI' 06, 2006.

[EQM96] El Emam, K.; Quintin, S.; Madhavji, N.H.: User Participation in the Requirements Engineering Process: An Empirical Study, Requirements Engineering, Volume 1, Issue 1, pp. 4-26, 1996.

[FFM04] Fogli, D.; Fresta, G.; Mussio, P.: On Electronic Annotation and its Implementation. In: Proceedings of the working conference on advanced visual interfaces, Gallipoli, Italy, pp. 98 – 102, 2004.

[HL01] Hofmann, H.F.; Lehner, F.: Requirements: Engineering as a Success Factor in Sofware Projects, IEEE Software Volume 19, Issue 4, Regensburg, Germany, pp. 58-66, 2001.

[Ji07] JING, http://www.jingproject.com/, viewed on 03.12.2007.

[KB03] Kensing, F.; Blomberg, J: Participatory Design: Issues and Concerns. In: Kensing, F.: Methods and Practices in Participatory Design. Copenhagen. ITU Press Copenhagen, Denmark, pp. 365-387, 2003.

[KC95] Keil, M.; Carmel, E.: Customer-Developer Links in Software Development, Communications if the ACM, Volume 38, Issue 5, pp. 43-51, 1995.

[KKL05] Kujala, S.; Kauppinnen, M.; Lehtola, L.; Kojo, T.: The Role of User Involvement in Requirements Quality and Project Success, IEEE International Conference on Requirements Engineering (RE'05), 2005.

[Ku05] Kujala, S.: User Involvement: A Review of Benefits and Challenges, Behavior & Information Technology, Volume 22, Issue 1, pp. 1-16, 2003.

[LDP05] Li, K.; Dewar, R.G.; Pooley, R.J.: Computer-Assisted and Customer Oriented Requirements Elicitation, Proceedings of the 13th IEEE International Conference on Requirements Engineering, Edinburgh, UK, 2005, pp. 479- 480, 2005.

[Ma96] Macaulay, L.: Requirements for Requirements Engineering Technique, Second International Conference on Requirements Engineering, (ICRE'96), Colorado Springs, USA, p. 157, 1996.

[Mo03] Moore, J.M.: Communicating Requirements Using End-User GUI Constructions with Argumentation, Proceedings of the 18th IEEE International Conference on Automated Software Engineering ASE'03, Montreal, Canada, pp. 360 – 363, 2003.

[Mo07] MORAE, http://www.techsmith.com/morae.asp, viewed on 03.12.2007

[NJ97] Nandhakumar, J.; Jones, M.: Designing in the Dark: the Changing User-Developer Relationship in Information Systems Development, Proc. ICIS, 1997.

[Po95] Potts, C.: Software Engineering Research Revisited, IEEE Software, Volume 10, Issue 5, pp. 19-28, 1995.

[RS92] Reeves, B.; Shipman, F.: Supporting Communication between Designers with Atrifact-Centred Evolving Information Spaces, Proceedings of the CSCQ '92, Toronto, Canada, pp. 394-401, 1992.

[SD06] Stevens, G.; Draxler, S.: Partizipation im Nutzungskontext. In: Heinecke, A.M.; Paul, H. (Eds.): Mensch & Computer 2006. Oldenbourg Verlag, Munic, Gemany, pp. 83- 92, 2006.

[Su93] Suchmann, L.: Forword. In: Schuler, D.; Namioka, A. (Hrsg.): Participatory Design: Principles and Practices. Lawrence Erlbaum, New York, pp. vii – ix, 1993.

Alignment of Software Specifications with Quality- and Business Goals in the SIKOSA Method

Andrea Herrmann[1], Daniel Weiß[2]

[1]Software Engineering Group, Faculty of Mathematics and Computer Science
University of Heidelberg
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
herrmann@informatik.uni-heidelberg.de

[2] Information Systems II (510 O)
University of Hohenheim
70593 Stuttgart, Germany
daniel.weiss@uni-hohenheim.de

Abstract: Business-IT alignment for software specifications means that the specifications have to be aligned with business goals. In the SIKOSA research project, we developed the SIKOSA method which supports the integrated assurance of quality during the whole software development process. In this work, we present these aspects of the SIKOSA method, which especially align specification decisions to quality goals and thus indirectly to business goals. Such goals play a role in the following activities: the derivation of software requirements from quality goals, the prioritization of these software requirements, and the definition of decision criteria for architectural design decisions. The results of all three activities influence architectural decisions.

1 Introduction

Business-IT alignment for software development means that the software (and all other artefacts) have to be designed in a way to support the business goals (*problem*). Assuming that the software works as it was specified, the software specifications have to be aligned with the business goals as well. Specifications are the result of a complex decision-making process which involves a variety of interdependent decisions on different levels of granularity, involving diverse stakeholders. Therefore, our objective in the research project SIKOSA was to develop a modelling method which supports a consistent alignment of specification-related decisions to business goals during different phases of the software development process (*object of investigation*) in an integrated way. For this purpose we took an overall *method engineering perspective*.

The SIKOSA method supports the integrated assurance of quality and of business-IT alignment during the whole software development process. No other software modelling method exists for doing so. Our prior work [HPK06] and [WKK07] describes this method. In the present work, we highlight those aspects of the SIKOSA method which

align specifications to quality- and business goals. The SIKOSA method consists of several modules. Those modules treating specification issues are: ProQAM (Process-oriented Questionnaires for Analyzing and Modeling Scenarios) [DOK05], TORE (Task Oriented Requirements Engineering) [PK03], MOQARE (Misuse-oriented Quality Requirements Engineering) [HP05], [HP07], and ICRAD (Integrated Conflict Resolution and Architectural Design) [HPP06].

Software properties and how well they are aligned with the business goals are defined by the decisions made during the software specification process. The requirements specification describes the needs, while the architectural design (specification) describes what will be implemented. Decisions based on the needs are made during the following activities: the software requirements specification, the definition of decision criteria for architectural design decisions and the prioritization of software requirements.[1] The results of all three activities influence the fourth activity: architectural decisions concerning the solution.

In the SIKOSA method, these activities produce the following artefacts (Figure 1):

1. Software requirements (here: MOQARE countermeasures) are derived from business goals by ProQAM and MOQARE.

2. Decision criteria for architectural design decisions with ICRAD are derived from business goals.

3. Software requirements priorities are attributed to the software requirements, taking into account the ICRAD decision criteria and the business goals.

4. Architectural design decisions are made with ICRAD.

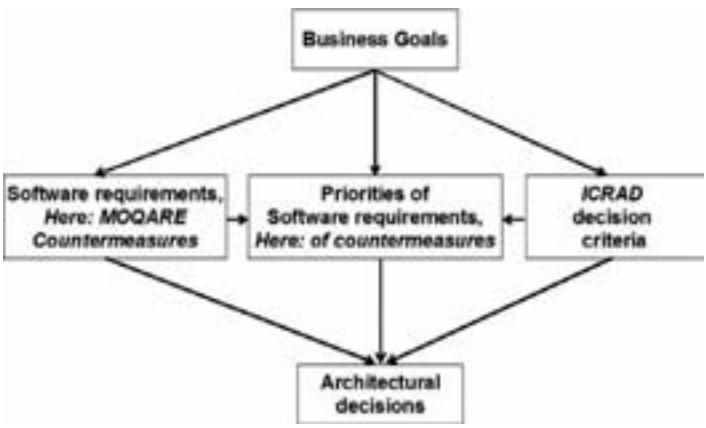

Figure 1: Business goals indirectly influence architectural decisions via three intermediate artefacts; the arrows signify relationships of the type "influences" between the artefacts

[1] Priorities support many further decisions such as conflict solution or test decisions.

The remainder of the paper is as follows: Section 2 cites related work. The subsequent sections treat the four above mentioned activities: Section 3 describes how countermeasures are derived from goals by MOQARE. Section 4 discusses ICRAD´s architectural design decision criteria, section 5 treats requirements prioritization. Section 6 presents how architectural decisions in ICRAD are indirectly aligned with goals, when they are based on software requirements, their priorities and the architectural decision criteria as defined in the preceding sections. Section 7 provides a summary.

2 Related Work

How decisions concerning specifications can be consistently aligned to business goals during different phases of the software development process, is no new question. Software modelling and specification methods have treated parts of this question, which now is fully treated by the SIKOSA method for the first time. In this section, we cite work which we built upon.

Business goals are "high-level reasons for getting the new product" [La02] and a "non-operational objective to be achieved by the […] system" [DVF93]. A lot of research activities focus on the business goals of software systems, projects or organizational units dealing with their classification and identification. Business goals can be categorized according to the five dimensions: product size, quality, staff, cost, and (calendar) time [Wi02]. Orthogonally to these dimensions, business goals can be classified according to the four perspectives of the Balanced Scorecard [KN92]: financial, customer, internal processes, learning & growth. For details, we refer the interested reader to the business literature mentioned above.

Software requirements and software requirements decisions can be described on different levels of granularity and with different focus. Aurum et al. [AWP06] distinguish four levels of requirements decisions: business, stakeholders, project, and product level. Lamsweerde et al. [La01] discern business goals, project goals, and software system goals. The distinction between business goals and software goals as well as their alignment are important features of the SIKOSA method.

The goal-oriented requirements engineering methods [La01] have been using software (product) goals successfully as a starting point for software requirements specification. In [He07], we have discussed how goal-orientation and hierarchical top-down detailing from goals to software requirements ideally supports decision-making during requirements elicitation. Other authors also emphasize the importance and multiple roles of goals for requirements elicitation, alignment of requirements with business goals, requirements validation, conflict solution and architectural design [YM98], [RS05]. However, these methods do not distinguish between business goals and software goals.

It seems logical to derive software requirements from business process requirements. Nevertheless, there are only few approaches to do so [BE01], [KL06a]. Business process modelling and software requirements modelling still use different notations and semantics. Approaches to their integration are presented by [SH00], [No04], [BCV05],

[KL06b]. However, some weaknesses of the integrations remain [BE01], [KL06a]. Especially, former work concentrates on functional requirements (FR). Non-functional requirements (NFR) are neglected, although they are gaining more and more relevance, as the competition on the market cannot be won by a software´s functional scope alone, but also quality is crucial. The SIKOSA method is the first one which models business process requirements as well as software requirements, FR as well as NFR.

Goals can serve as decision criteria. This means that among several available alternatives, the one is chosen which supports the goal(s) best. Which goals and decision criteria are used in a specific context depends on the stakeholder preferences. However, in literature it is not discussed which further factors influence the choice of a decision criterion. During our literature research and case study experience, we found that the decision criterion strongly depends on the question which is to be answered during a specific software development phase [He07]. For instance, requirements engineering aims at identifying those requirements which are most beneficial to the stakeholders, while architectural design chooses that design which satisfies the requirements best. Furthermore, the satisfaction of some criteria can not be estimated in each phase, e.g. reliable cost estimates are more difficult to obtain during requirements engineering than during design, when there is some – even preliminary – knowledge about the IT system´s realization. This is why in the SIKOSA method use different decision criteria for each of the four activities shown in Fig. 1.

3 Derivation of software requirements from software goals

The distinction between business goals and software goals is fundamental in the SIKOSA method. Software goals can be functional or non-functional goals. In the SIKOSA method, the functional goals are described by the business processes to be supported, while the non-functional goals are called quality goals.

We integrate the ProQAM business requirements modelling with software requirements specification based on quality goals and countermeasures (NFR described with MOQARE) and use cases (FR described with TORE). Usually, goal-oriented analyses proceed from high-level goals down to requirements [He07]. This is supported systematically by the modules of the SIKOSA method, as presented in Figure 2.

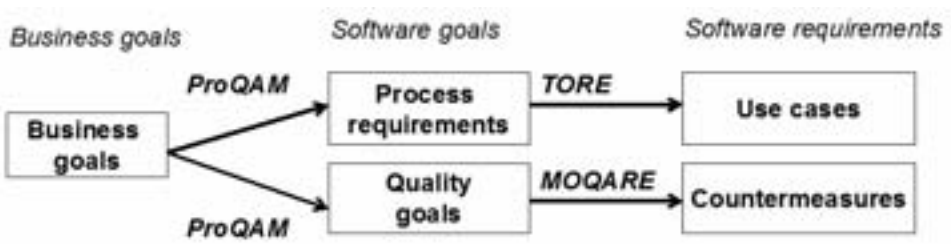

Figure 2: ProQAM, TORE and MOQARE derive software requirements from business goals

The five concepts shown in Figure 2 are defined as follows:

- Business goal: ProQAM identifies the stakeholders´ business goals. These can be formal or technical and express situations to be achieved and results respectively modes of action by means of decisions.

- Process requirements: The process requirements describe the process which is to be executed. Not only does it contain the steps which are to be supported by software, but also staff needed or relevant competences. In ProQAM, such elements are described by central constructs of event-driven process chains (EPCs [Sc01]). The central element, the function, is defined in a way to support the business goals.

- Use case: Use cases [Co01] describe the requirements for the interaction between user (or other, maybe non-human actors) and the software, including pre-conditions, interaction steps and post-conditions. They can be derived from the process requirements. Deriving FR from business processes in the form of such use cases is supported by the method TORE.

- Quality goal: A quality goal is a goal which is to be satisfied by the software and therefore is a high-level NFR. In MOQARE, quality goals are expressed by the combination of an asset plus a quality attribute, like "usability of the user interface". An asset can be any protectable part of the system. A quality attribute describes an aspect or characteristic of quality.

- Countermeasure: A countermeasure is an operational requirement which supports the quality goal. Countermeasures can be FR, exception scenarios of use cases, NFR constraining use cases, architectural constraints, user interface constraints, constraints on project and software development, constraints on administration or maintenance, or another quality goal.

The five concepts above describe desired properties of the business, the business processes, and the software. From the security field, the idea of negative, undesirable concepts has been adopted in the SIKOSA method. The most famous concept based on this principle is the misuse case [SO00], [SO01], [Al02]. Like use cases, misuse cases describe the interaction of the software system with an actor, but misuse cases describe unwanted scenarios (e.g. attacks, user errors, accidents) which threaten goal satisfaction. Misuse cases help to define and to complement the software requirements and also to document the justification of these requirements. This principle is used in the SIKOSA method with respect to business goals and quality goals.

Due to limited space, we focus on the realization of Business IT-alignment in the SIKOSA method. For a complete description of the methods, we refer the reader to the publications cited in the introduction. In the remainder of this section, we describe how

countermeasures are derived from functional and non-functional process requirements by MOQARE. These process requirements are output of ProQAM[2].

To illustrate our methods, we describe a case study performed during the Sysiphus enhancement[3]. Sysiphus is a tool which is developed and used at the University of Heidelberg and the Technical University of Munich to teach software engineering and to document the results of case studies [Sy07]. Sysiphus implements TORE and MOQARE. It also supports design according to Brügge and Dutoit [BD04] and ICRAD. The case study objectives were: We wanted to test and to measure the usability of Sysiphus and to propose requirements on potential improvements. These requirements had to be prioritized in order to be integrated into plans for the further enhancement of Sysiphus. Finally, a workshop was held to discuss strategies of how to implement the improvements and a decision was made.

To meet these objectives, this case study included the following steps:

1. Definition of a usage context and the business goal
2. Description of the FR
3. Detailing of the quality goal „usability of the user interface“ and derivation of countermeasures, in order to define what usability means for this system
4. Benefit estimation for the FR and countermeasures
5. Usability test and evaluation of the software to measure how well the countermeasures and the quality goal “usability” are satisfied
6. Prioritization of the countermeasures for release planning
7. Decision on implementation alternatives

The results of the steps 4 to 6 are presented in section 5, and step 7 in section 6.

Step 1: We restricted the scope of the analysis to the requirements engineering (RE) and architectural design (AD) modules of Sysiphus. Their business goal is “efficient support of RE and AD”. The analysis started with the quality goal “usability of user interface”, which in a former analysis (not presented here) had been identified to contribute to this business goal. We assumed a usage context where Sysiphus is applied in a small company by ten IT professionals. They are irregular users, had only short Sysiphus training and are offered no helpline support. They must use the tool during RE and AD.

Step 2: The FR supported by the RE & AD part of Sysiphus are described by 27 use cases, such as “specification of misuse cases” or “review of design”.

---

[2] We want to remark that one of the strengths of the SIKOSA method is that it integrates modular methods which can be applied independently of the others as well as in combination.

[3] Further MOQARE case studies have been published here: [HRP06], [HKD07], [HP07]. However, most industry case studies we performed are confidential.

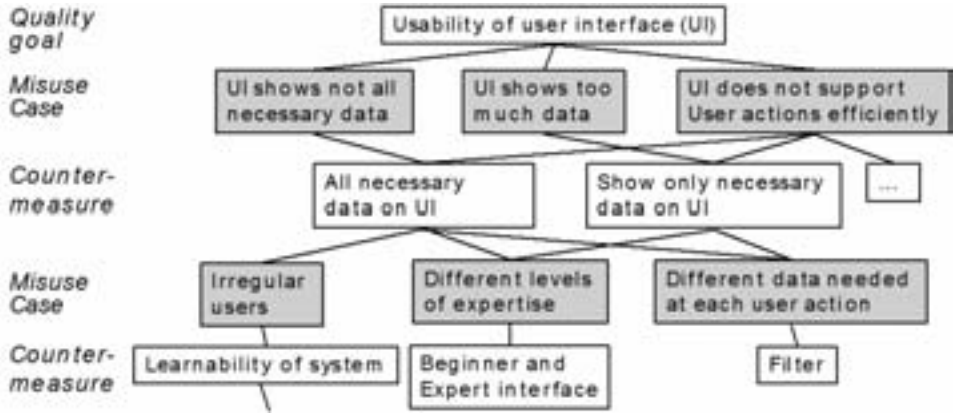

Figure 3: Section of the Misuse Tree resulting from the case study

Step 3: From quality goals, MOQARE derives misuse cases and countermeasures. The misuse cases threaten the quality goals. A countermeasure reduces the probability of a misuse case or reduces its predicted negative consequences. By analyzing the quality goal with MOQARE, 22 misuse cases and 31 countermeasures were identified. Two ISO standards [ISO13], [ISO92] supported the identification of usability requirements, which then were chosen and detailed specifically to the context and its needs. Figure 3 shows a section of the resulting Misuse Tree. System specific misuse cases and countermeasures should be worded in a way to apply to the 27 use cases individually, but we did not do so here because so many details would have complicated the Misuse Tree and all later treatments of the countermeasures.

4 ICRAD decision criteria for architectural decisions

As business-IT alignment is our objective, the decision criteria for architectural decisions have to be defined in a way to support the business goals. The business goals usually can not be used as design decision criteria directly. For instance, it might be difficult to estimate how well an architectural solution supports the business goal “high market share” or “efficient process support”, as their satisfaction does not depend on the software alone. It is easier to predict how well the quality goal “usability of user interface” is supported.

Frequently used decision criteria for architectural decisions are benefit, cost, complexity and risks, or combinations of these factors, like net value and benefit-cost-ratio [XMC04] [KAK01], [IKO01]. Therefore, in ICRAD these are the four standard evaluation criteria for architectural alternatives (see section 6). If necessary, ICRAD can be adapted in order to use other or additional criteria, like the satisfaction of goals [GY01], of non-functional goals [KAK01], [IKO01] or of functional goals [CB95], [KAB96]. But usually, if the benefit of these goals is known, their satisfaction is taken into account by considering their contribution to the benefit.

5 Prioritization of requirements

In [HPP06], we discussed that some requirements conflicts can only be solved knowing the possible technical solutions and by selecting one of these architectural alternatives. However, one out of three types of requirements conflicts can be solved without this knowledge, based on requirement priorities [HPP06]. Presorting of requirements is useful for supporting such conflict solution and other decisions. Davis talks of "requirements triage" [Da03] and also the "Planning Game" of Extreme Programming [Be00] classifies requirements according to which ones have to be implemented, which can be postponed and which have to be analyzed in more detail. Such a classification facilitates decisions like release planning. During architectural decisions, must-requirements can be an exclusion criterion: Those architectural alternatives which do not satisfy the must-requirements will not be considered further. Some decisions later in the software development process can use requirement priorities, like testing (where requirements priorities support test case prioritization) or – as in the case study – the assessment of the overall level of quality.

Step 3 of the case study identified requirements which Sysiphus should satisfy in order to support the quality goal "usability of user interface". Some of these countermeasures are currently not satisfied, while others are (at least partly) satisfied by Sysiphus. In the case study, we used two prioritization criteria: With respect to the overall assessment of the usability of Sysiphus, our main criterion for requirements prioritization was its benefit relative to the usability quality goal. For the planning of later software releases, it was important whether and how well a countermeasure is already satisfied; its implementation cost also played a role.

In MOQARE, we derive a countermeasure´s benefit from the risk reduction which it causes with respect to the misuse case risk. Misuse Case risk is defined as the product of probability and caused damage [ISO02], [XMC04]. Common methods for requirements prioritization[4] do not consider dependencies among the benefits of requirements at all or only superficially. In reality, however, such dependencies are frequent and critical. For instance, countermeasures can replace each other partly, when they mitigate the same misuse case. Or countermeasures may need each other for being effective against the same misuse case. We take into account such dependencies by bundling requirements and by relating all estimations to a reference system [HP06]. In many prioritization methods, it is common to bundle those requirements which depend on each other most in relatively independent bundles[5]. The reference system is the idea of a set of requirements which are imagined to be implemented. If perfect quality is the benchmark, the perfect system is the reference, i.e. a system in which all requirements are implemented

[4] Such methods are the analytic hierarchy process (AHP) [Sa80], [KWR98], numeral assignment [Ka96] or cumulative voting (CV), also called "$100 test" [LW00], [BJ06]. According to [HP06], all methods which attribute one fixed priority value to each requirement can be said to neglect dependencies.

[5] These groups are then called features [RHN01], [Wi99], feature groups [RHN01], super-requirements [Da03], classes of requirements [REP03], bundles of requirements [PSR04], categories [XMC04], User Story [Be00], super attributes [SKK97] or Minimum Marketable Features [DC03].

[XMC04]. The reference system can also be the ensemble of all mandatory requirements [REP03], the former system version or a competitor´s product.

When estimating a countermeasure´s benefit relative to a reference system, the risk of the corresponding misuse case(s) is estimated twice: Firstly, the "reference risk" in the reference system is estimated, secondly the "varied risk" if this countermeasure is not implemented or if it is implemented additionally. The benefit achieved by a countermeasure in relation to a misuse case equals the risk reduction [AH04], [XMC04].

On this basis, we can continue with the case study´s *Step 4:* The reference system was defined to support all the FR identified in step 2 plus all countermeasures defined in step 3. This means that our benchmark is the system with perfect usability. The benefit of this reference system is set to 100 benefit points. This benefit is defined to be achieved by the satisfaction of the FR alone. Then, the satisfaction of the usability goal does not add direct benefit, but only prevents risk. We use the unit "benefit points", because it is difficult in an example with fictitious usage context to estimate benefits in Euro.

The FR benefits were defined on two levels of granularity. On a high level, we identified three FR bundles defined according to the three methods supported. We assumed these bundles to be independent and simply distributed the 100 benefit points. On the use case level, within each bundle use case benefits were estimated.

Misuse case probabilities were estimated in percentage and damages in benefit points relative to the total system benefit of 100. Resulting benefits for the most important and some less important countermeasures are shown in Table 1. There usually are n-m-relationships among misuse cases and countermeasures, which complicate the estimations. Countermeasures which need each other for being effective, should be bundled and estimated like one. All others are estimated individually relative to the reference system. We here discuss the countermeasure "all necessary data on user interface" (which is a quality goal itself and further analyzed in Figure 3). It refers to two misuse cases. In the reference system, the risk of both misuse cases is supposed to be 0. If the countermeasure was not implemented, then the user – as a workaround - can open several Sysiphus windows and this way get all necessary data. However, this does not work for all user actions and it is inefficient. The misuse case "User interface does not show all necessary data" causes a damage of 100 points, because it makes the system useless. However, this happens only in an estimated 40% of the user actions. Therefore, its varied risk without the countermeasure being implemented is 40 points. Without the countermeasure, the other misuse case – "the user interface does not support the user efficiently" - is true to 100%. As the users are obliged to use the system and because they are IT professionals, who can handle two windows on their screen, the damage was estimated to be only 10 points (the value of loss of productive work time). Assuming that both misuse cases are independent of each other, the countermeasure´s benefit then is 40+10=50 benefit points. As can be seen in the table, all other countermeasures have a much lower benefit. There was no other misuse case in the analysis which caused such high damage.

Table 1: Countermeasure benefits resulting from the case study (Remark: These benefits are specific to the case study and not generally valid.)

Benefit (in benefit points)	Countermeasure
50	All data necessary for one user action must be presented at the same time.
2.0	At any time, the currently executed user action must be obvious to the user.
1.8	The system allows filtering of data.
1.4	Automated check whether input data are within the valid range
1.1	Context sensitive help for any screen and data field
1.02	User training
1.0	Success notification after completion of each user action
1.0	Support of users for doing the user actions in the right order
1.0	Explanations on user interface + self-explanatory names
…	
0.1	The system allows to adapt the size of the work space.
0.0875	Data fields are initialized with default values.

Step 5: Usability test: To save time during the case study, we did not specify detailed test cases for evaluating the current satisfaction of the usability requirements by the system. Instead, we executed the 27 use cases as defined in step 2 and assessed how well each of them satisfies each of the countermeasures. The results of these tests were entered in a spreadsheet table where each column corresponds to a use case and each row to a countermeasure. These results x_{ij} measure the degree of satisfaction of a countermeasure i during the execution of a use case j between 0 (not satisfied at all) and 1 (perfectly satisfied). These tests were performed by two testers and the results were discussed afterwards to obtain a shared judgement.

The satisfaction of each countermeasure i was calculated as weighted sum $x_i = \Sigma_j (x_{ij} \cdot$ *benefit* of use case j). If all countermeasures were satisfied, the total system benefit would have been 100 points. As some were only partly satisfied, the total usability risk (benefit loss) was the weighted sum = Σ_i [(1- x_i) · (*benefit* of countermeasure i)]. This risk was 18 points and consequently the effective benefit of the system 100-18 = 82[6]. This value will be especially interesting when we will re-assess the usability after a system enhancement to measure the usability improvement quantitatively.

Step 6: Countermeasure prioritization for release planning: For those countermeasures which are not yet satisfied to 100%, the cost of doing so was estimated in 1, 2 or 3 cost points. The priority of a countermeasure i with respect to release planning was defined to be proportional to "(1-x_i) · *benefit* of countermeasure i". Those

[6] We must remark here that we were very strict when evaluating the software!

countermeasures with the highest priorities and those with cost = 1 were candidates to be scheduled for the next release.

6 Architectural decisions which are aligned with goals

ICRAD [HPP06] is an iterative and integrated process for the solution of requirements conflicts and for architectural design. In this section, we describe how it compares architectural alternatives and how the decision is made. Decisions among two or more alternatives and their justifications are documented in the template shown in Table 2. Each alternative is evaluated with respect to its benefit, risk, implementation cost and complexity cost. The reference system can be different for each decision, as it is modified by the decisions made before[7]. The *benefit* of an alternative is not equal to the sum of the benefits of the requirements realized by this alternative, due to dependencies. The *risk* of an alternative includes risks provoked by realizing risky requirements or provoked by the architectural alternative, as well as risks provoked by not realizing some countermeasures. *Cost* of implementation ideally is estimated in the same unit as the benefit, in order to be comparable. *Complexity* includes architectural and organizational complexity and will lead to maintenance and other cost. For being comparable to the other criteria, complexity is transformed into complexity cost. Complexity is caused by software complexity, e.g. by coupling of its components [KAB96], [CB95], [LRV99] and also by the complexity of the software´s integration into its environment. These estimations are done for both (respectively all) alternatives of the same decision and their results are documented in Table 2.

Table 2: Template table used to compare alternatives in ICRAD.

	Alternative 1	**Alternative 2**	**Difference**
Cost	C1	C2	C2-C1
Complexity Cost	CC1	CC2	CC2-CC1
Risk	R1	R2	R2-R1
Benefit	B1	B2	B2-B1
Total benefit	B1-R1	B2-R2	(B2-R2)-(B1-R1)
Total cost	C1+CC1	C2+CC2	(CC2- CC1)+(C2-C1)
Net value	(B1-R1)- (C1+CC1)	(B2-R2)- (C2+CC2)	(B2-R2)-(C2+CC2) -(B1-R1) +(C1+CC1)
Total Benefit/ total cost	(B1-R1) / (C1+CC1)	(B2-R2) / (C2+CC2)	[(B2-R2)-(B1-R1)] / [(CC2- CC1)+(C2-C1)]

[7] This – together with requirements dependencies – is why the requirements benefits estimated in section 4 cannot be re-used directly here.

The *total benefit* is calculated as *benefit* minus *risk*. *Total cost* includes implementation plus complexity *cost*. Two decision criteria are:

- net value = *total benefit* minus *total cost*
- Benefit-cost-ratio = *total benefit* / *total cost*

If the more expensive solution has a lower benefit, then it is logical to choose the cheaper and better solution. However, very often, the alternative with the higher benefit is the more expensive one, as is also the case in our case study. The value *[(B2-R2)-(B1-R1)] / [(CC2- CC1)+(C2-C1)]= ΔTB/ ΔTC* (see Table 2, in the lower right field) has shown to be a good third decision criterion [HPP06]. These three criteria do not always lead to the same decision. How to proceed if they are in favour of different decisions is described in [HPP06].

In the case study, we have identified a multitude of countermeasures which signify improvement ideas. One might have realized them in a series of subsequent releases improving the user interface´s usability incrementally. As an alternative, we considered re-designing the user interface. This decision was fundamental and was discussed in a workshop of several hours with ten participants. The workshop started with a discussion of the countermeasures and architectural alternatives. Without going into detail, we want to present the resulting decision between two alternatives. Although in the preceding steps, a countermeasure´s benefit was a main criterion for its prioritization, now the default criteria of ICRAD have all been taken into account, because the implementation cost, complexity cost and risks caused by a solution also play a role for the decision for or against the one or the other solution. The must-requirement "All data necessary for one user action must be presented at the same time." (from Table 1) was realized in both alternatives. The other countermeasures were considered indirectly by estimating the benefit on the basis of which countermeasures can be realized by each of the alternatives. The benefit of a requirement was again measured in "benefit points", relative to the 100 value of the perfect system. Benefit should ideally be comparable to cost, yet in this case study they were not. The cost of each alternative here was estimated in person months (unlike cost estimation for individual requirements in step 6).

Table 3: Comparison of alternatives in the case study; "PM" stands for "person months" and "BP" for "benefit points"

	Alternative 1: incremental improvement	**Alternative 2: re-design**	**Difference (Alternative 2 – Alternative 1)**
Cost	3 PM	6 PM	3 PM
Complexity Cost	2 PM	1 PM	-1 PM
Risk	0 BP	2 BP	2 BP
Benefit	6 BP	14 BP	8 BP
Total benefit	6 BP	12 BP	6 BP
Total cost	5 PM	7 PM	2 PM
Net value	6 BP – 5 PM	12 BP – 7 PM	6 BP – 2 PM
Total Benefit/	1.20 BP/ PM	1.71 BP/ PM	3 BP/ PM

As can be seen in Table 3, the re-design has higher implementation cost than the incremental improvement, but lower complexity cost because it reduces the software´s complexity[8]. It can be expected that the re-design achieves a much higher improvement of the usability and, therefore, more benefit, but also includes the risk to loose benefit. The re-design has the higher total cost and higher total benefit. The net values are difficult to compare, as cost and benefit are estimated in different metrics. The benefit-cost-ratio is higher for the re-design. Criterion $\Delta TB/\Delta TC$ (right bottom field) also is in favor of the re-design. Therefore, the re-design was chosen.

7 Summary

This work presents in which way the SIKOSA method aligns software specification and decisions to functional as well as to non-functional quality- and business goals. The following four activities align specification with goals and, therefore, are presented here: the software requirements specification, the prioritization of these software requirements, the definition of decision criteria for architectural design decisions, and making architectural decisions. For aligning software specification and software with goals consistently, it is important to execute these four activities in an integrated way, as it is done by the SIKOSA method, unlike in any other method.

The four activities were executed in a case study where software usability requirements were defined, the usability was assessed, the most important improvements identified, and finally a decision was made between two alternatives: incremental improvement and re-design of the user interface. This quantitative decision-support has shown to be a good artefact for structuring discussions, documenting decision rationale and identifying missing information. However, estimating and consensus-making among several stakeholders demands more time compared to ad hoc decisions. Therefore, we recommend applying such approaches mainly to such decisions which have an important impact and/ or are difficult to make.

References

[AH04] Arora, A.; Hall, D.; Pinto, C.A.; Ramsey, D.; Telang, R.: An ounce of prevention vs. a pound of cure: How can we measure the value of IT security solutions? Lawrence Berkeley National Laboratory. Paper LBNL-54549. 2004.

[Al02] Alexander, I.: Misuse Cases Help to Elict Non-Functional Requirements. http://easyweb.easynet.co.uk/~iany/consultancy/ misuse_cases/misuse_cases.htm

[AWP06] Aurum, A.; Wohlin, C.; Porter, A.: Aligning Software Project Decisions: A Case Study. In: Int. J. of Software Eng. and Knowledge Eng., 16(6), 2006, pp. 795-818.

[8] We remark that the next release was defined in a way that at realistic cost a good improvement could be attained without any risk. A larger release would have caused more cost without significantly higher usability improvement.

[BCV05] Bleistein, S.J.; Cox, K.; Verner, J.: Strategic Alignment in Requirements Analysis for Organizational IT: an Integrated Approach. In: ACM Symposium on Applied Computing, Santa Fe, 2005.

[BD04] Bruegge, B.; Dutoit, A.H.: Object-Oriented Software Engineering - Using UML, Patterns, and Java, Prentice Hall, 2004.

[Be00] Beck, K.: Extreme programming explained, Addison-Wesley, Upper Saddle River, 2000.

[BE01] Brücher, H.; Endl, R.: Erweiterung von UML zur geschäftsregelorientierten Prozessmodellierung. In: Proc. Referenzmodellierung RefMod2001, http://www.wi.uni-muenster.de/is/Tagung/Ref2001/Kurzbeitrag13.pdf

[BJ06] Berander, P.; Jönsson, P.: Hierarchical Cumulative Voting (HCV) - Prioritization of Requirements in Hierarchies. In: Int. J. of Software Eng. and Knowledge Eng., 16(6), 2006, pp. 819-849.

[CB95] Clements, P.; Bass, L.; Kazman, R.; Abowd, G.: Predicting Software Quality by Architectural-Level Evaluation. In: Proc. 5th Int. Conf. on Software Quality ICSQ, Maribor, Slovenia, 1995.

[Co01] Cockburn, A.: Writing effective use cases, Addison-Wesley, 2001.

[Da03] Davis, A.M.: The Art of Requirements Triage. In: IEEE Computer 36(3) 2003, pp.42-49.

[DC03] Denne, M.; Cleland-Huang, J.: Software by Numbers: Low-Risk, High-Return Development. Prentice-Hall, 2003.

[DOK05] Dietrich, A.; Otto, S.; Kirn, S.: Simulationsmodell für logistische Prozesse in Mass-Customization-Szenarien. In: Kirn et al. (Hrsg.): Kundenzentrierte Wertschöpfung mit Mass Customization. 2005, pp. 118-147.

[DVF93] Dardenne, A.; van Lamsweerde, A.; Fickas, S.: Goal-Directed Requirements Acquisition. In: Science of Computer Programming 20, 1993, pp. 3-50.

[GY01] Gross, F.; Yu, E.: Evolving system architecture to meet changing business goals: An agent and goal-oriented approach. In: Proc. Fifth IEEE Int. Symposium on Requirements Engineering, 2001, pp.316 - 317.

[He07] Herrmann, A.: Entscheidungen bei der Erfassung nicht-funktionaler Anforderungen. Workshop "Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung", SE 2007, Hamburg, Germany, 2007.

[HKD07] Herrmann, A.; Kerkow, D.; Doerr, J.: Exploring the Characteristics of NFR Methods – a Dialogue about two Approaches. In: Proc. 13th Int. Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality – REFSQ 07, Trondheim, Springer, 2007; pp. 320-334.

[HP05] Herrmann, A.; Paech, B.: Quality Misuse. In: Proc. 11th Int. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 05, Essener Informatik Beiträge, Band 10, 2005; pp. 193-199.

[HP06] Herrmann, A.; Paech, B.: Benefit Estimation of Requirements Based on a Utility Function. In: Proc. 12th Int. Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 06, Essener Informatik Beiträge, Band 11, 2006; pp.249-250.

[HP07] Herrmann, A.; Paech, B.: MOQARE: Misuse-oriented Quality Requirements Engineering. In: Requirements Engineering Journal, to be published.

[HPK06] Herrmann, A.; Paech, B.; Kirn, S.; Kossmann, D.; Müller, G.; Binnig, C.; Gilliot, M.; Illes, T.; Lowis, L.; Weiß, D.: Durchgängige Qualität von Unternehmenssoftware. In: Industrie Management, 6, 2006; pp. 59-61.

[HPP06] Herrmann, A.; Paech, B.; Plaza, D.: ICRAD: An Integrated Process for Requirements Conflict Solution and Architectural Design. In: Int. J. of Software Eng. and Knowledge Eng. 16(6) Dec. 2006, pp. 917-950.

[HRP06] Herrmann, A.; Rückert, J.; Paech, B.: Exploring the Interoperability of Web Services using MOQARE. IS-TSPQ Workshop “First International Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems”, Bordeaux, 2006.

[IKO01] In, H.; Kazman, R.; Olson, D.: From Requirements Negotiation to Software Architectural Decisions. In: Proc. from Software Requirements to Architectures Workshop STRAW, 2001.
[ISO02] International Standards Organization ISO: Risk management – Vocabulary – Guidelines for use in standards. ISO Guide 73, International Standards Organization, 2002.
[ISO13] International Standards Organization ISO: Norm DIN EN ISO 13407, Benutzer-orientierte Gestaltung interaktiver Systeme.
[ISO92] International Standards Organization ISO: Norm DIN EN ISO 9241, Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten.
[KAK01] Kazman, R.; Asundi, J.; Klein, M.: Quantifying the Cost and Benefits of Architectural Decisions. In: Proc. Int. Conf. Software Eng., 2001; pp.297-306.
[KAB96] Kazman, R.; Abowd, G.; Bass, L.; Clements, P.: Scenario-based analysis of software architecture. In: IEEE Software, 13(6), 1996; pp. 47-55.
[Ka96] Karlsson, J.: Software requirements prioritisation. In: Proc. 2nd Int. Conf. Requirements Engineering, 1996; pp.110-116.
[KN92] Kaplan, R.S.; Norton, D.P.: The Balanced Scorecard: Measures That Drive Performance. In: Harvard Business Review, 70(1), 1992; pp. 71-79.
[KL06a] Korherr, B.; List, B.: Aligning Business Processes and Software Connecting the UML 2 Profile for Event Driven Process Chains with Use Cases and Components. In: Proc. 18th Int. Conf. on Advanced Inf. Systems Eng. CAiSE´06, Luxembourg, 2006: pp. 19-22.
[KL06b] Korherr, B.; List, B.: A UML 2 Profile for Event Driven Process Chains. A UML 2 Profile for Business Process Modelling. In: Proc. 1st Int. Workshop on Best Practices of UML at the 24th Int. Conf. on Conceptual Modeling ER 2005, Klagenfurt 2005.
[KWR98] Karlsson, J.; Wohlin, C.; Regnell, B.: An evaluation of methods for prioritizing software requirements. In: Information and Software Technology 39, 1998; pp. 939-947.
[La01] van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proc. of 5th Int. Symposium on Requirements Eng., 2001; pp. 249-263.
[La02] Lauesen, S.: Software Requirements - Styles and Techniques. Addison-Wesley, 2002.
[LRV99] Lassing, N.; Rijsenbrij, D.; van Vliet, H.: On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn´t Everything. In: Proc. Second Nordic Software Architecture Workshop NOSA 99, 1999; pp. 1103-1581.
[Lu00] Luftman, J.N.: Assessing business/IT alignment maturity. In:Comm. of AIS, 4(14), 2000.
[LW00] Leffingwell, D.; Widrig, D.: Managing Software Requirements - A Unified Approach, Addison-Wesley, Reading, Massachusetts, USA, 2000.
[No04] Noran, O.S.: Business Modelling: UML vs. IDEF. Griffith University, School of Computing and Information Technology, 2004.
http://www.cit.gu.edu.au/~noran/Docs/UMLvsIDEF.pdf (last visit: 12 nov 2007)
[PK03] Paech, B.; Kohler, K.: Task-driven Requirements in object-oriented Development. In (Leite, J.; Doorn, J., eds.): Perspectives on Requirements Engineering, Kluwer Academic Publishers, 2003.
[PSR04] Papadacci, E.; Salinesi, C.; Rolland, C.: Payoff Analysis in Goal-Oriented Requirements Engineering. In: Proc. 10th Int. Workshop on Requirements Eng.: Foundation of Software Quality – REFSQ04, 2004.
[REP03] Ruhe, G.; Eberlein, A.; Pfahl, D.: Trade-Off Analysis For Requirements Selection. In: Int. J. of Software Eng. and Knowledge Eng. 13 (4), 2003; pp. 345-366.
[RHN01] Regnell, B.; Höst, M.; Natt och Dag, J.; Beremark, P.; Hjelm, T.: An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. In: Requirements Eng. 6, 2001; pp. 51–62.
[RS05] Rolland, C.; Salinesi, C.: Modeling Goals and Reasoning with Them. In (Aurum, A.; Wohlin, C., Eds.): Engineering and Managing Software Requirements, Springer, Berlin, Heidelberg, 2005.
[Sa80] Saaty, T.L.: The Analytic Hierarchy Process, McGraw-Hill, New York, 1980.

[Sc01] Scheer, A.-W.: ARIS – Modellierungsmethoden, Metamodelle, Anwendungen. 4th edition, Springer, Berlin, 2001.

[SH00] Scheer, A.; Habermann, F.: Making ERP a success. In: Communications of the ACM, 43(3), 2000; pp. 57-61.

[SKK97] Stylianou, A.C.; Kumar, R.L.; Khouja, M.J.: A total quality management-based systems development process. In: ACM SIGMIS Database, 28(3), June 1997, pp. 59-71.

[SO00] Sindre, G.; Opdahl, A.L.: Eliciting Security Requirements by Misuse Cases. In: TOOLS Pacific, 2000; pp. 120-131.

[SO01] Sindre, G.; Opdahl, A.L.: Templates for Misuse Case Description. In: Proc. 7th Int. Workshop on Requirements Eng.: Foundation of Software Quality – REFSQ 01, Essener Informatik Beiträge Band 6, 2001; pp. 125-136.

[So01] Sommerville, I.: Software Engineering, Pearson Education Deutschland, 6th ed. 2001.

[Sy07] Sysiphus http://sysiphus.in.tum.de/, 2007 (last visit: 12 nov 2007)

[Wi99] Wiegers, K.E.: First things first: prioritizing requirements. In: Software Development 7(9), September 1999.

[Wi02] Wiegers, K.E.: Success Criteria Breed Success. In: The Rational Edge, 2(2), 2002.

[WKK07] Weiß, D.; Kaack, J.; Kirn, S.; Gilliot, M.; Lowis, L.; Müller, G.; Herrmann, A.; Binnig, C.; Illes, T.; Paech, B.; Kossmann, D.: Die SIKOSA-Methodik – Unterstützung der industriellen Softwareproduktion durch methodisch integrierte Softwareentwicklungsprozesse. In: Wirtschaftsinformatik 49(3), 2007; pp. 188-198.

[XMC04] Xie, N.; Mead, N.R.; Chen, P.; Dean, M.; Lopez, L.; Ojoko-Adams, D.; Osman, H.: SQUARE Project: Cost/Benefit Analysis Framework for Information Security Improvement Projects in Small Companies. Technical Note CMU/SEI-2004-TN-045, Software Engineering Institute, Carnegie Mellon University, 2004.

[YM98] Yu, E.; Mylopoulos, J.: Why Goal-Oriented Requirements Engineering? In: Proc. 4th Int. Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality – REFSQ 1998, Pisa, Presses Universitaires de Namur, 1998, pp. 15-22

Extended Privacy Definition Tool

Martin Kähmer and Maike Gilliot
{kaehmer, gilliot}@iig.uni-freiburg.de

Abstract: Eliciting non-functional security requirements within a company was one of the major aspects of the SIKOSA project[1]. Scenarios, such as that of METRO presented in this paper, show how besides a company's internal requirements, customers' preferences also play an important role. However, conflicts between specific customers' privacy policies and those of a company need to be detected and dealt with. To this end we present a policy language that is able to tackle this comparison problem and two tools: An editor tool allowing users to specify their policies in a user-friendly way and a monitoring tool to evaluate und enforce the policies at runtime.

1 Introduction

Personalized services are often viewed as the panacea of e-commerce [SSA06]. User profiles, such as click streams logging which sites users access, are used to generate a profile of the interests of the users. The decisive advantage of such services lies in the opportunity of entering a one-to-one relationship in order to achieve more effective customer loyalty. Service tailoring is thereby no longer limited to e-commerce. In Germany, the METRO-Group is developing the "Future Store", where shopping trolleys are fitted with personal shopping assistants, i.e. computers connected to the store's information system. Services, such as recommender systems, are personalized by means of customer cards [KA06]. However, personalization involves intensive collection and usage of personal-related data. If customers want to benefit from such services, enforcing privacy by minimizing data disclosure is no longer possible. Customers need means to control the usage of their data [PHB06].

In this paper, we present the **Ex**tended **P**rivacy **D**efinition **T**ool (ExPDT) as a means for companies to comply not only with regulatory and business requirements, but also with customers' privacy preferences. In Section 2, we classify our ExPDT solution consisting of a policy language and corresponding tools. The language itself is described in Section 3, and the editor tool and the monitor tool are presented in Section 4. The closing section gives an outlook on further work.

[1] SIKOSA: Sichere Kollaborative Softwareentwicklung und Anwendung [WKK+07], in collaboration with the Uni of Heidelberg, Uni of Hohenheim, and ETH Zürich, funded by MWK of the Land Baden-Württemberg

2 Tackling the privacy problem

Privacy and security for enterprise information systems is about ensuring that business processes are executed as expected and operations such as data accesses are in accordance with a prescribed or agreed on set of norms, such as laws, regulations, and decisions. Solutions to achieve this can be broken down to two main approaches according to the time of application. One approach is called retrospective reporting, where traditional audits usually done through manual checks based on comprehensive logs and reports of the last period of time are used to show policy conformance [Acc08].

The other, more recent approach is often called security by-design, exhibiting a more preventive focus. Non-functional privacy and security requirements are captured and subsequently propagated into the enterprise applications. We propose a model of different layers with respect to abstraction and potential for automation (cf. Figure 1). Since laws only describe what has to be done in general, these regulations have to be interpreted to obtain control objectives for the particular business domain of a company. Although formulated by experts, these control objectives are still in natural language. For IT systems, they need to be interpreted once more and mapped to the particular services, components, and employees of the company. Policies are a set of formalized rules specifying precisely for each unit what is allowed or mandatory and what is prohibited. Such policies serve as input for security monitors, enforcing them on the lower system level. A high degree of automation is only possible within the policy and the monitor layer, where ExPDT is situated, as this is the first level providing laws in a machine-readable format.

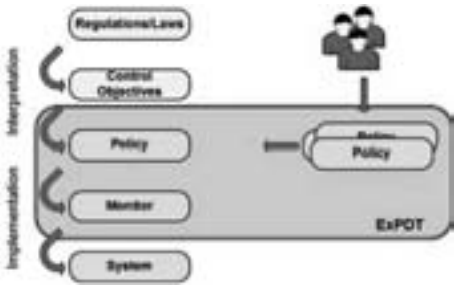

Figure 1: ExPDT within the layer model

2.1 Policy and monitor requirements

For a policy language, it is not merely essential to feature sufficient expressiveness based on a wide range of compassing modalities like permissions, prohibitions and orders as well as on context inclusion based on a fine grained vocabulary [HPSW06, BAKK05]. It is also necessary for a policy language to allow for modular specification and policy comparison so that every single requirement can be addressed and combined with a valid policy for deployment and, since requirements can stem from different regulations and privacy agreements, inconsistencies or contradictions are taken into consideration. Conflicting rules result in operational risk and should be detected and as far as possible solved. Not just regulatory objectives that could be realized in one central policy have to be enforced by a company. Its customers need to be able to control the collection and usage of their own personal data by formulating their own privacy and security preferences [Bun83, PHB06].

To enforce such expressive languages, monitors not only have to cope with conditions, i.e. "traditional" access control, but also control orders and obligations. As those types of rules are generally not enforceable, the monitor has to provide other means to control the fulfillment on system level.

2.2 Related Work

Tackling privacy by means of policies is not new. The World Wide Web Consortium (W3C) developed the Platform for Privacy Preferences (P3P) to express privacy policies in a machine-readable format and its counterpart, the P3P exchange language APPEL, to express customers' preferences [W3C06, CLM05]. Both lack conditions, obligations, and any kind of enforceability and are thus drastically reduced in their usage control capability. For internal policies, the eXtensible Access Control Markup Language XACML [Mos05] was designed by the OASIS consortium as open standard to specify expressive policies that can be interpreted and enforced by a security monitor. XACML even provides policy combination tools to support distributed policies, although it is not suitable for comparing policies, because the intersection of two general policies is not defined. The WS-Policy framework [WSP07] for web services provides a general purpose model and syntax to describe and communicate policies of web services, which consists of sets of different kinds of assertions, e.g. for security, privacy or reliability. Although allowing for optional assertions, this flexibility cannot be guided by sanctions or penalties. While XACML and WS-Policy can be used for privacy related policies, IBM's Enterprise Privacy Authorization Language (EPAL) is dedicated to this task [AHK$^+$03]. It accounts for the further usage of accessed data objects by supporting obligation elements in its policy rules and exhibits a fine grained vocabulary as well as monitor integration. The Novel Algebraic Privacy Specification (NAPS) framework enhances EPAL on a logical level to an algebra additionally allowing for modular specification of policies and adds a concept of sanctions to allow for flexible rule adherence [RS06]. However, common to all of these policy languages is the lack of adequate operators for comparing and analyzing policies.

3 ExPDT – Expressing Privacy Policies

The Extended Privacy Definition Tool (ExPDT) language allows users to specify declarative privacy and security policies over specific domain knowledge using OWL-DL, a computational complete and decidable subset of the OWL Web Ontology Language [MvH04] corresponding to Description Logic. The ExPDT language is used to describe permissions, prohibitions, and orders that have to be adhered to if certain contextual conditions are met or some obligations have to be fulfilled. As ExPDT is geared towards dynamic environments, it deals with incomplete context information and also includes sanctions that can be imposed. Based on the algebraic framework NAPS, it inherits semantics and combination operators allowing for a modular specification of policies. A distinguishing feature of the language is the difference operator for policy comparison. For deployment

of the language, an editor tool was developed. Additionally, a tool for interpreting and enforcing ExPDT policies is presented in Section 4.

For the presentation of the ExPDT language in the following chapters, we introduce three logical layers: the language layer, the domain layer and the instance layer. At the bottom, the language layer establishes the basis by providing fixed vocabulary for the specification of language itself, just like the grammar of a natural language. Based upon that, the domain layer fills up vocabulary by defining the instances of assets, actuators and environment. In contrast to the language layer, the specifications on the domain layer have to be consistent with the current scenario. Therefore, they are subject to occasional adaptations. A common understanding of privacy preferences is not possible, until language and domain are commonly defined. On the third layer, concrete policy instances both of the companies as well as of the customers can be defined, exchanged and agreed upon.

3.1 Language specification layer

A language is made from its syntax and semantics. Hereby, syntax is the definition of all words allowed to be used in the language as well as their set up, in particular the definition of a rule and its parts. The semantics describes the meaning behind the syntax. For a policy, the semantics is given by its evaluation function that provides the results for a particular policy query.

3.1.1 Syntax

Although the ExPDT language features a representation in OWL, its syntax is presented in a more space-saving way on the basis of the simplified OWL class diagram with the inheritance and selected properties of the OWL classes shown in Figure 2. Short examples of the actual OWL syntax are given in Sections 3.2 and 3.3 for domain and instance layer.

A *policy* consists of a prioritized list of rules and a default ruling in the case where no rule applies. A *rule* itself is comprised of one or more possibly negated guards constraining the scope of this rule from users, actions, data and purpose, a number of conditions and the ruling that subsequently delivers the decision of this rule. Hence, a generic rule has the following form: $[(\mathrm{user}, \mathrm{data}, \mathrm{action}, \mathrm{purpose}), \mathrm{conditions}, \mathrm{ruling}]$.

For intuitive specification of the scenario on the domain layer later on, the element instances of a *guard* are partially ordered in hierarchical structures allowing for grouping of instances and the formulation of policies rules applying to entire sub-hierarchies, e.g. to all users of a particular department or all the data belonging to contact information. Thereby, each of them has his own structure: customers, employees and services are combined in the *user* structure, sensitive data items are described in *data*, possible actions on these items in *action* and the possible intentions of actions in question are structured in *purpose*.

Regulations often depend on context information, e.g. permitting data access only if the customer is not under age or the legal guardian has given his consent. For the inclusion of such constraints, ExPDT reverts to a 3-valued, many-sorted condition logic. A condition

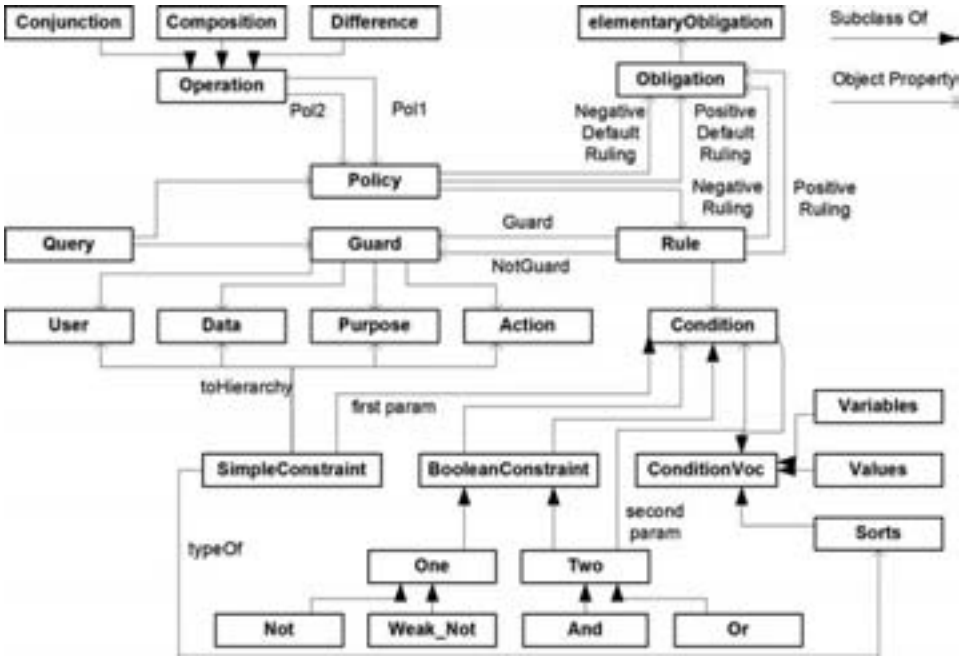

Figure 2: Syntax of ExPDT policy language as class diagram

is a formula of this condition logic defined over the condition *vocabulary* and its interpretation functions. A vocabulary consists of the final set of *sorts* (i.e. variable types) each with a final set of *variables*. The set of non-logical symbols of *simple constraints* includes relations, the set of logical symbols the operators *and*, *or*, *not*, *weak not*, 0, 1 and u as undefined. The single-valued operators not and weak not have only a first parameter, the others a second additional one. Formulas and terms of the condition logic are recursively defined as usual as in the predicate logic free of quantors. The semantics of a formula is given as in the 3-values Łukasiewicz L3 logic. The undefined value is advantageous to an environment of dynamic character, such as stores with continuously changing customers and modified or switched services, in that it supports the rule evaluation even with incomplete context information as will be shown in Section 3.1.2.

A policy rule not only regulates the actions on data items, but can impose *obligations*, such as "notify customer" or "delete data within one day". In contrast to many other policy languages, ExPDT does not consider obligations as pure black box instructions, but has an underlying obligation model of a half lattice above the power set of the *elementary obligations* $\tilde{O}$, subset as relation, conjunction as aggregation, with maximum element top $\top$ as the empty obligation and the minimal element bottom $\bot$ as the impossible obligation. Imposing the obligation $\top$ means that the action of the guard can be carried out without further undertaking, imposing $\bot$ that an action may not be carried out. Eliminations of contradicting elementary obligation combinations, such as "delete data within a week"

Modality	Obligations	Sanctions	Ruling
Permission			$(\top, \top)$
	o^+		$(o^+, \top)$
Prohibition			$(\bot, \top)$
		o^-	$(\bot, o^-)$
Order			$(\top, \bot)$
	o^+		$(o^+, \bot)$
		o^-	$(\top, o^-)$
	o^+	o^-	(o^+, o^-)
Error			$(\bot, \bot)$

Table 1: ExPDT codes modalities into ruling.

and "keep data for a year" at the same time, can be achieved by excluding from the lattice all those obligation sets containing problematic obligations.

The *ruling* of an ExPDT rule and the default ruling of the overall policy are specified by a tuple of obligation (postiveObligation, negativeObligation). Since ExPDT policies make statements about users performing an action on some data for a particular purpose, the policy query is accordingly also a tuple of user, action, data, and purpose, too.

3.1.2 Semantics

While the specification of policies, queries and rulings were part of the syntax, the evaluation function of a query resulting in a ruling for a given policy defines the semantics of the ExPDT policy language. Firstly, the semantics of a single ruling is explained, followed by the description of the evaluation function.

Ruling of a rule The required authorization and order rule modalities presented can both be expressed in the ExPDT language, as shown in Table 1. Actions can therefore not only be permitted but also forbidden. It is also possible to compulsorily regulate the execution of actions. In addition to the conditions obligations can also be imposed on the user. These are actions that have to be performed in future. The ExPDT language also allows the users a certain degree of freedom in rule compliance. While this always applies in the case of a permit – if an action is allowed, one does not necessarily have to use this right – users can decide for themselves whether they adhere to a prohibition or a command. If they do not, sanctions in the form of additional obligations can be specified in an ExPDT rule. If these sanctions correspond however to the impossible obligation , adherence becomes necessary for the users. The various rule modalities are mapped in ExPDT via the tuple of the ruling. Here are some examples:

- Permission: A retailer can access the customer number. Ruling: ($\top$, $\top$)
- Permission with obligation: A retailer can access the customer's shopping list but not secretly. Ruling: (notify, $\top$)

- Prohibition with sanction: A retailer may not access the shopping list, which is achieved by imposing the impossible obligation. Disregarding this prohibition, he must inform the customer and pay a fine as sanction. Ruling: ($\perp$, payFine)
- Compulsory command: The security administrator must in any event classify the data requested according to its sensitivity. The sanction according to the impossible obligation makes adherence to this order indispensable. Ruling: ($\top$, $\perp$)

Evaluation of a policy The semantics of the policy language is determined through the evaluation function $eval_\alpha(\mathcal{P}, q)$ for a query q regarding a particular policy $\mathcal{P}$ and current assignment α of the contextual condition variables. Roughly, the function searches through the list of policy rules until a rule is matched by the query. Matching means that all elements of the rule guard are either equal to the user, action, data, and purpose of the query or stand higher up in their corresponding hierarchy. Additionally, the condition of the rule must not evaluate to false using the current variable assignment. Thus, queries are not restricted to minimal elements of the guard hierarchies and allow for scenarios, where, for example, a basic policy company policy referring to departments is only composed with a department policy making concrete statements about individuals. Although the particular user Bob may not be mentioned in the company policy, its rules still apply to him. The complete evaluation algorithm works as follows:

1. Initialize the ruling r_p with ($\top$, $\top$) and preset evaluation status v to default.
2. Evaluate rules one by one according to their priority.
    a. If the rule's guard is matched by the query and its condition evaluates to 1, return the conjunction of the rule's ruling and hitherto accumulated r_p as policy ruling and an evaluation status v of final.
    b. If the query is matched by a rule's guard and its condition evaluates to u, add rule's ruling to r_p, set the status v to applicable and proceed with the next rule.
3. If the status v is applicable, than return r_p as ruling and that status.
4. If the status is still default, no rule has matched and the default ruling is returned together with the status v default.

The case of incomplete context information resulting in an undefined condition value for a rule is taken into account by accumulating the ruling of such a rule with a possibly previous found ruling, i.e. conjunct both the positive obligations and the negative obligations, and proceeding with the evaluation. Hence, it is ensured that the evaluated ruling is possibly too restrictive due to the additional obligations, but never too weak.

Combination of policies The extensive dragging along of the evaluation status v with its distinction of final, applicable or default ruling allows ExPDT the definition combination operators despite the stub-behavior of the policies. The stub-behavior corresponds to the

intention of the default ruling is to ensure a safe ruling until another rule matches, therefore the refinement of a default ruling with an applicable or final one should be possible in case of a policy combination. In ExPDT, two combination operators are defined: the conjunction $\mathcal{P}_1 \wedge \mathcal{P}_2$ thereby evaluates $\mathcal{P}_1$ and $\mathcal{P}_2$ with equal priority, while the composition $\mathcal{P}_1||\mathcal{P}_2$ gives $\mathcal{P}_1$ higher priority for the evaluation. For more detailed combination tables of the rulings, generating algorithms, and algebraic laws see [Rau04].

Comparison of policies In related literature, for the comparison of two guidelines one often finds the equivalence where both guidelines always supply the same results and the refinement which examines a guideline as to whether it is more restrictive or specific than another. In practice, these tests are however only suitable to a certain extent for users, if they can only determine with them whether their preferences are fulfilled by a service. How should the users behave, however, if the policy of a service does not correspond to that of their own, therefore being not equivalent or more restrictive? They will not reject a utilization of the service in each case. It can even be their wish, depending on the current situation, to lower their data protection demands in favor of the utilization.

In such situations, the users must be able to quickly survey and estimate how far the service guideline deviates from their own preferences or the service's previous ruling. In dynamic environments, in particular, where the users are faced with many different services and their respective individual policies, this task can no longer be manually accomplished. In order to alleviate the user's personal decision for or against service utilization, the difference operator for two guidelines is defined in the following. This operator reduces the regulation of the policy to become effective to those rules describing situations that are of interest to the users for their assessments, namely to those allowing additional actions or at least actions on weaker conditions or obligations and so supply more generous results.

Difference: Given two policies $\mathcal{P}_1$ and $\mathcal{P}_2$ over compatible vocabulary, the difference $\mathcal{P}_2 - \mathcal{P}_1$ is a mapping from $\mathcal{P} \in \mathcal{P}$ to a list of rules R that covers exactly those queries q and assignments α of conditional variables that result in a less restrictive ruling for $\mathcal{P}_2$, so $(r_i, v_i) = eval_\alpha(\mathcal{P}_i, q)$ for $i = 1, 2$ and $r_1 \not\leq r_2$. For these, the difference rule list results in the same decisions as $\mathcal{P}_2$.

This rule list describes the functional difference of both policies, so they are compared independent of their possible evaluation status; the stub behavior of the policies is not taken into consideration. This is particularly significant if policy $\mathcal{P}_1$ is to be replaced by a different policy $\mathcal{P}_2$, for example if a customer discards his own preferences $\mathcal{P}_1$ and releases his personal data under the service's policy $\mathcal{P}_2$. Then it is irrelevant whether an action is forbidden owing to the standard ruling of $\mathcal{P}_1$, but this standard ruling is refined with a permit. It is only important here that this action is subsequently permitted. However, if policies $\mathcal{P}_1$ and $\mathcal{P}_2$ are intended to be connected afterwards, the difference should consider the stub-behavior of $\mathcal{P}_1$. For instance, the $\mathcal{P}_1$ default ruling can be replaced by an arbitrary non-default ruling of $\mathcal{P}_2$, which would provide a more specific result without the need to get the users' attention – as long as they are aware of this stub-behavior. In fact, the former mentioned equivalence and refinement of two policies can be computed by means of the

difference: if $\mathcal{P}_2 - \mathcal{P}_1$ results in an empty list, $\mathcal{P}_2$ describe less restrictive situations and $\mathcal{P}_2$ is a functional refinement of $\mathcal{P}_1$. If the difference of switched policies results in an empty rule list as well, $\mathcal{P}_1$ and $\mathcal{P}_2$ are functionally equivalent.

An initial implementation can take place here by way of a brute force approach. The decisions for all possible enquiries and all possible allocations of the environment parameter must simply be calculated for both $\mathcal{P}_1$ and $\mathcal{P}_2$ policies. A rule with the scope of the query (i.e. corresponding guard) and the conditions and ruling of the rule appropriate in each case of $\mathcal{P}_2$ is included in the rule list $\mathcal{P}_2 - \mathcal{P}_1$, if the ruling r_1 has other or lesser obligations than r_2. This brute force approach tests all possible combinations of enquiries and parameter allocations so that its complexity grows exponentially with the vocabulary used.

Algorithm 1 difference(R_1, R_2): walking through the rule sets R_1 and R_2

```
 1: PolicyDIFF := ∅
 2: for i:= max(R2) downto min(R2) do
 3:    for j:= max(R1) downto min(R1) do
 4:       RuleDIFF := rulecomp(R1[j], R2[i])
 5:       if RuleDIFF ≠ ∅ then
 6:          for all rd in RuleDIFF do
 7:             if guardOf(rd) ∧ ¬guardOf(R1[j]) ≠ ∅ then
 8:                PolicyDIFF += differenz( (R1[j − 1] downto R1[max]), rd )
 9:             else
10:                PolicyDIFF += rd
11:             end if
12:          end for
13:          next i
14:       end if
15:    end for
16: end for
17: return PolicyDIFF
```

Therefore, a more efficient approach is presented in this paper. As outlined by Algorithm 1, the rules of both policies are looped through according to their priority, so that each rule of $\mathcal{P}_2$ is compared with all rules of $\mathcal{P}_1$. According to the guard logic, guards can contain disjunctions, conjunctions and negations of guards. Therefore, guards cannot be compared using their top-elements alone, but by the set of hierarchy elements described by them, defined by the closure. For an element h, the closure $\overline{h}$ consists of h itself and all elements lower in the hierarchy. The guard operators $\vee$, $\wedge$, and $\neg$ can be mapped accordingly $\cup$, $\cap$ and $\not\leq$. If such a comparison detects only equal or more restrictive situations with bigger scope or weaker conditions and obligations, the looping is continued with the remaining rules of $\mathcal{P}_1$. If there are, however, such situations and if they are not captured by a following $\mathcal{P}_1$-rule with lower priority (cf. recursive call), they are formally captured by a new rule that is appended to the difference result. Then, the looping is discontinued for the current $\mathcal{P}_2$ rule and starts with the next rule anew. If all $\mathcal{P}_2$ rules are examined, the construction of the difference terminates. However, before Algorithm 1 can start, the policies have to be preprocessed by upgrading the default rulings and normalizing $\mathcal{P}_2$.

For the construction of the functional difference, a distinction between normal rules and default ruling is not necessary. Hence, the standard rulings of both policies are upgraded to

Algorithm 2 DiffNorm(R): Normalization of rule list

$R^N := \emptyset$
for i:= $max(R)$ downto $min(R)$ {rules in order of priority} **do**
  (g, c, r) := split i {splitting of a rule in its parts}
  M := $\emptyset$
  for all $f \in \{0,1\}^{|l_g|}$ **do**
    g_f:=g
    c_f:=c
    for $n = 1$ to $|l_g|$ **do**
      if $f(n) = 0$ **then**
        $c_f := c_f \wedge l_c$
      end if
      if $f(n) = 1$ **then**
        $g_f := g_f \wedge l_g$
      end if
      $M := M \cup (g_f, c_f, r)$
    end for
  end for
  $R^N := R^N \cup M$
  $l_c := l_c \cup \neg c$
  $l_g := l_g \cup \neg g$
end for
return R^N

normal rules by appending rules to the list with the root elements of the guard hierarchies, without conditions, and with default ruling as rule ruling: (user_{root}, data_{root}, action_{root}, $\mathrm{purpose}_{root}$), 1, (default ruling). If there is more than one root for a hierarchy, append a rule for each of them. These rules match all possible queries by design, so that the default ruling is not triggered anymore and can be disregarded for the difference construction.

Since the evaluation function of ExPDT considers the policy rules as a prioritized list with dependencies, the rules of $\mathcal{P}_2$ cannot be individually compared; $\mathcal{P}_2$ has to be normalized first. Following Algorithm 2, for each rule all the situations matched by rules with higher priority are explicitly excluded from its guard and conditions. For $\mathcal{P}_1$, this normalization is not necessary because the recursive call already copes with the dependencies.

For the comparison of two rules in Algorithm 3, it has to be determined whether there is an overlap between the two scopes. If they do not overlap, the comparison stops. Otherwise, the difference between these two rules is examined by the following case differentiation:

- $r_1 \not\geqslant r_2$: The ruling of $rule_2$ is less restrictive or different. Independent of the conditions, $rule_2$ is appended to the rule difference and returned.
- Otherwise: The ruling r_2 is stricter or equal. Hence, up to two rules have to be appended to the rule difference:
  - For queries matching $guard_2$ but not $guard_1$, this stricter or other ruling is in any case new, so that a rule with these queries as guard, conditions of c_2, and ruling r_2 is appended.
  - For queries also matching $guard_2$ as $guard_1$ (i.e. the disjunction of their closures), the r_1 is less restrictive, but a less restrictive condition c_2 can neces-

Algorithm 3 rulecomp($rule_1, rule_2$): comparison of two rules

```
1:  (g1, c1, r1) := split rule1
2:  (g2, c2, r2) := split rule2
3:  RuleDIFF := ∅
4:  if g2 ∧ g1 ≠ ∅ then
5:    if r1 ⋡ r2 then
6:      RuleDIFF += (g2, c2, r2)
7:    else
8:      if g2 ∧ ¬g1 ≠ ∅ then
9:        RuleDIFF += ((g2 ∧ ¬g1), c2, r2)
10:     end if
11:     if c2 ↛ c1 then
12:       RuleDIFF += ((g2 ∧ g1), (c2 ∧ ¬c1), r2)
13:     end if
14:   end if
15: end if
16: return RuleDIFF
```

sitate another difference rule. This new rule should only describe the new situations, so that its condition is $c_2 \wedge \neg c_1$.

Example: $c_1 = (\leq 18)$, $c_2 = (\leq 18 \vee GuardianOK)$
$c_2 \wedge \neg c_1 = (\not\leq 18 \wedge GuardianOK)$

For the last case, it is essential to not only evaluate conditions but to also compare the ability to satisfy two given conditions. It must be determined whether one rule restricts its applicability with more strict or equivalent conditions than another rule or features a greater application space with additional context situations. This yields in the examination whether a condition c_1 satisfies another c_2, i.e. if c_1 is satisfied, then c_2 is also satisfied, or if c_1 results in the undefined state u, c_2 is also undefined or even satisfied. For independency of the current situation, this has to hold for all possible variable assignments. This examination is, however, NP-complete over the number of variables of the vocabulary considered, so that no efficient algorithm is to be expected for the general case. Nevertheless, in order to be able to compare the conditions, the examination is reduced to a satisfy relation similar to [BKBS04], which is at least correct, i.e. if two conditions c_1 and c_2 are contained in the relation, then the above-mentioned satisfaction holds true.

Satisfy relation: Given a conditional vocabulary $\mathfrak{Voc}$, the satisfy relation is the relation $\rightarrow_{\mathfrak{Voc}} \subseteq C(\mathfrak{Voc}) \times C(\mathfrak{Voc})$. The relation is correct if for all conditions $c_1, c_2 \in C(\mathfrak{Voc})$ for all possible assignments α holds (in infix notation):

$$c_1, c_2 \in \rightarrow_{\mathfrak{Voc}} :\Rightarrow \Big[eval_\alpha(c_1) = true \Rightarrow eval_\alpha(c_2) = true\Big] \vee \Big[eval_\alpha(c_1) = \emptyset \Rightarrow \big(eval_\alpha(c_2) = true \vee \emptyset\big)\Big]$$

If the opposite direction also holds, the satisfy relation is complete. A correct satisfy relation can often be constructed via the symbolic evaluation by all pairs of atomic formulas

```
<owl:Class rdf:about="DATA:shoppingList">
  <rdfs:subClassOf rdf:about="DATA"/>
</owl:Class>

<owl:Class rdf:about="DATA:Prescription">
  <rdfs:subClassOf
    rdf:resource="DATA:shoppingList"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
        rdf:resource="COND:contains"/>
      <owl:someValuesFrom
        rdf:resource="DATA:drugs"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<Prescription rdf:ID="DATA:shopAtDrugstore"/>
```

Listing 1: Definition of guard elements

```
<owl:Class rdf:about="OBL:Delete">
  <rdfs:subClassOf
    rdf:resource="OBL:ELEOBLIG"/>
  <owl:disjointWith>
    <owl:Class rdf:about="OBL:keep"/>
  </owl:disjointWith>
</owl:Class>

<Obligation rdf:about="OBL:delete_Notify">
  <rdfs:subClassOf
    rdf:resource="OBL:keep"/>
  <rdfs:subClassOf
    rdf:resource="OBL:notify"/>
</Obligation>
```

Listing 2: Definition of obligations

with known semantics satisfy dependency, but also via the comparison on a pure syntactic level of their interpretation functions of the same sort. Such a correct satisfy relation is mostly adequate for practical application, even if it is not complete. For if two conditions are mutually dependent and this dependency is unknown to the users and therefore not included in the relation, then such conditions should be independently treated in order to meet the users' expectation. At worst, the differential result hereby increases by an additional rule, however without changing its evaluation.

3.2 Domain layer

Based upon the language layer, the vocabulary is filled up by defining concrete instances of assets, actuators, conditions, and obligations and by categorizing them accordingly on the domain layer. These specifications should always be consistent with the current scenario and, therefore, need to be adapted in case of environmental changes. ExPDT uses an ontology specified in OWL-DL, as it supports not only for the representing of domain specific knowledge, but also for the automated interpretation and reasoning.

Listing 1 gives an example of the data hierarchy specification. Here, a shopping list is a subclass of the root DATA. In turn, a medical prescription is a subclass of a shopping list, and its instances are dynamically assigned by a property restriction: all shopping lists that contain at least one drug to buy are considered as a prescription. Hence, customers can formulate rules for this particular kind of shopping list, taking care not to provide its contents to normal salespersons, but only to the druggist of the shop. Examples of obligation are given in Listing 2. The obligation "delete data after usage" is coded as an elementary obligation that, of course, is incompatible with the obligation to keep the data afterwards. Obligations used in rulings are either instances of single or multiple elementary obligations.

3.3 Policy layer

On the policy layer, instances of policy can be formulized by combining the building blocks of the domain layer stuck together with the elements of the language layer. Listing 3 shows a privacy policy containing only two rules. The first rule allows a druggist to access a particular prescription for t he purpose of giving further information, e.g. the compatibleness of some substances. The second rule allows all persons working in sales to read shopping lists, if the customer has consented, the data is deleted afterwards and the customer is notified about this event. Queries for all other situations are not matched by the rules, but by the restrictive default ruling of the policy that prohibits everything not already covered by the rules. As in the policy instance, the assignment of the conditional variables to the guard hierarchies – the customer has not given consent for actions involving his data – is part of the policy layer, but the actual variable values have to be provided by the monitor.

```
<POL:Policy rdf:ID="MyPolicy">
  <POL:PolicyHasRules>
    <rdf:Seq rdf:ID="MyPolicyRules">
      <rdf:li>
        <POL:Rule rdf:ID="Rule1">
          <POL:RuleHasUser rdf:resource="USER:druggist"/>
          <POL:RuleHasAction rdf:resource="ACTION:read"/>
          <POL:RuleHasData rdf:resource="DATA:shopAtDrugstore"/>
          <POL:RuleHasPurpose rdf:resource="PURP:information"/>
          <POL:RuleHasPosObligation rdf:resource="OBL:top"/>
          <POL:RuleHasNegObligation rdf:resource="OBL:top"/>
        </POL:Rule>
      </rdf:li>
      <rdf:li>
        <POL:Rule rdf:ID="Rule2">
          <POL:RuleHasUser rdf:resource="USER:sales"/>
          <POL:RuleHasAction rdf:resource="ACTION:read"/>
          <POL:RuleHasData rdf:resource="DATA:shoppingList"/>
          <POL:RuleHasPurpose rdf:resource="PURP:advertisment"/>
          <POL:RuleHasCondition rdf:resource="COND:hasConsented"/>
          <POL:RuleHasPosObligation rdf:resource="OBL:deleteAndNotify"/>
          <POL:RuleHasNegObligation rdf:resource="OBL:top"/>
        </POL:Rule>
      </rdf:li>
    </rdf:Seq>
  </POL:PolicyHasRules>
  <POL:PolicyHasPosDefaultRuling rdf:resource="OBL:bottom"/>
  <POL:PolicyHasNegDefaultRuling rdf:resource="OBL:top"/>
</POL:Policy>

<ASS:Data rdf:about="DATA:Customer">
  <ASS:gaveConsent rdf:resource="Consent:no"/>
</ASS:Data>
```

Listing 3: Example of policy instance with two rules

4 ExPDT – The tools

The previous section described the semantic and syntactic aspects of the policy language itself. In the following we focus on the tools making the language usable. We will first

Figure 3: Displaying policies and specifying operations using the ExPDT policy editor

present an editor tool as graphical user interface for users to define their privacy preferences, followed by the components of a monitor tool for the evaluation and the enforcement of policies on the system layer.

4.1 Editor tool

Basically, any generic OWL editor can be used as graphical user interface for ExPDT, as it enables the user to handle the various OWL-DL files, such as domain knowledge and policy instances, to visualize and to edit the contained classes and their properties. However, the limited usability of those editors prevents its deployment for laymen. Hence, we are developing a lightweight editor with reduced capabilities that hides the loquaciousness of OWL-DL. As seen in Figure 3, this editor not only allows for the editing of ExPDT policies, but can also be used for requesting policy operations, such as comparison or query evaluation, and subsequently displaying of the results returned by the monitor tool.

4.2 Monitor tool

An access control monitor prevents unacceptable behavior by running alongside an untrusted program and intercepting all actions that would lead to undesirable states [HMS06]. Access control rules without obligations and sanctions are always enforceable, as well as some obligations. Starting from the architecture of the ExPDT monitor in Figure 4, we present the main components of the monitor and their interplay. Whenever the Policy Enforcement Point receives a request, it is forwarded to the Policy Decision Point, which selects the “relevant” rules from the policy and evaluates it by dint of the Policy Information Point. The access decision is then returned to the Policy Enforcement Point, which grants or denies the access.

Policy Decision Point: The guard's elements user, action, data, and purpose are mapped to their corresponding hierarchies in order to determine the relevant policy rules for a given query. As the rules are ordered by priority, only the first matching rule is considered and evaluated. Further rules matching the guard only come into play, if the conditions statement returns the value "undefined". ExPDT handles an undefined condition values as true, but imposes on the access further obligations by accumulating the obligations of following matching rules.

Figure 4: Components of ExPDT monitor tool

Policy Information Point: The Policy Information Point assigns the values to the conditions. Conditions either relate to former accesses or events or to the system environment. Some conditions are static: Once the value of the condition "over 18" is true, it stays true. In contrast, the value of the condition "has consented" has to be evaluated at each request, as the user may have in between revoked his consent. To this end, a mapping from the language ontology onto real system events is required. For example, a message reporting a click on a button in the GUI has to be interpreted as "user has consented". The mapping of the received events onto the ontology is done mainly manually. Approaches to handle this translation automatically are presented for example in [GMP06].

Policy Enforcement Point: The enforcement component has to impose the access decision of the evaluation component onto the system. This is easy for policy rules without obligations or sanctions. They are enforceable, as an access control monitor is able to evaluate each request and to prohibit policy violating accesses. Enforcement of obligations and sanctions in general is not possible, as both prescribe actions a subject has to perform some time in the future. An online monitor has thus no possibility to enforce that the obliged action is accomplished within the given timeframe. However, the monitor is able to supervise the events and to report violations or fulfillment *after* the period allotted. An online monitor can evaluate the obligation "user has to be informed about data usage within 2 days" by checking after the two days if the user notification has been sent. However, obligations expressing a prohibition can be enforced if they can be transferred into preconditions. Examples are the Chinese Wall policy rules, where a user reading some confidential data about company A is in future not allowed to read confidential data of competing company B. This obligation can be expressed as precondition to further access requests and is thus enforceable.

For the evaluation of ExPDT policies, a prototype was implemented in Java. This tool provides for the functionality of a policy management point by offering the difference operator for policy comparison and the combination operators. (cf. Section 3.1.2)For acting as a policy decision point, it can evaluate policy requests given an assignment of environment variables. The prototype is based on the Jena OWL framework [Hew07] that not only allows for parsing the OWL files, but also provides an API to reason about the con-

tained information. As the internal reasoner of Jena is still quite limited for OWL-DL, the external Pellet reasoner [Cla08] is attached. It evaluates queries in description logic (DL) on the given knowledge base and relieves ExPDT of the computation of class inheritances and memberships, and the reconstruction of the full guard hierarchies.

5 Summary and conclusion and further steps

Due to technologies like RFID, personalized services are not longer limited to the e-commerce world. Motivated by an example of the retailer METRO, we show that enterprises can provide personalized services to their clients based on the personal data collected during their shop visits. Customers accept the release of personal data if they are still able to control the usage of their data. Thus, enterprises have - beside their internal security requirements - also to respect their customers' privacy preferences and demonstrate compliance to their policies. This requires an expressive policy language and a mechanism to prove compliance to the customer and regulatory bodies. This paper focuses on the policy language. We present the extended privacy definition tool ExPDT expressing privacy preferences for access and usage of personal data, focusing on the comparison and merging features of the language. We further describe the implementation of the ExPDT enforcement monitor describing which ExPDT rules can be enforced, and which cannot. The proposed tools contribute to the compliance by design approach.

For now, ExPDT enables differences between policies to be detected. How these differences can be resolved has not yet been considered. The next step, therefore, is a negotiation protocol. The goal is not a fully automated procedure, but a tool to assist the negotiation process step by step. Enforcement of obligations is a open research issue. We are currently investigating how obligations can be enforced by rewriting business process. A second approach to the enforcement of obligations uses heuristics in order to determine at runtime process executions that will probably lead to obligation violation.

References

[Acc08] Rafael Accorsi. Privacy Audits to Complement the Notion of Control for Identity Management. In de Leeuw, Fischer-Hübner, Tseng, and Borking, editors, *Policies and Research in Identity Management*, volume 261. Springer, 2008.

[AHK+03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise Privacy Authorization Language (EPAL 1.2). Submission to W3C, 2003.

[BAKK05] Travis D. Breaux, Annie I. Antón, Clare-Marie Karat, and John Karat. Enforceability vs. Accountability in Electronic Policies. Technical Report TR-2005-47, North Carolina State University Computer Science, 2005.

[BKBS04] Michael Backes, Günter Karjoth, Walid Bagga, and Matthias Schunter. Efficient comparison of enterprise privacy policies. In *Proc. of 2004 ACM Symposium on Applied Computing*, pages 375 – 382, 2004.

[Bun83] Bundesverfassungsgericht. Volkszählungsurteil. In *Entscheidungen des Bundesverfassungsgerichts*, volume 65. 1983. Urteil 15.12.83; Az.: 1 BvR 209/83; NJW 84, 419.

[Cla08] Clark & Parsia. Pellet OWL-Reasoner. `http://pellet.owldl.com`, 2008.

[CLM05] Lorrie Faith Cranor, Marc Langheinrich, and Massimo Marchiori. A P3P Preference Exchange Language 1.0 (APPEL). Technical report, W3C, 2005.

[GMP06] Christopher Giblin, Samuel Müller, and Birgit Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. Technical report, IBM Research Zürich, 2006.

[Hew07] Hewlett-Packard Development Company. Jena: A Semantic Web Framework for Java. `http://jena.sourceforge.net/`, 2007.

[HMS06] Kevin W. Halmen, Greg Morrisett, and Fred B. Schneider. Computability Classes for Enforcement Mechanisms. *ACM Transactions on Programming Languages and Systems*, 28:1, 2006.

[HPSW06] Manuel Hilty, Alexander Pretschner, Christian Schaefer, and Thomas Walter. Enforcement for Usage Control - A System Model and an Obligation Language for Distributed Usage Control. Technical Report I-ST-18, DoCoMo Euro-Labs Internal, 2006.

[KA06] Martin Kähmer and Rafael Accorsi. Kundenkarten in hochdynamischen Systemen. In *Proc. of KiVS NETSEC 2007*, 2006.

[Mos05] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report, OASIS, 2005.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language – Overview. W3C Recommendation `http://www.w3.org/TR/owl-features/`, 2004.

[PHB06] Alexander Pretschner, Manuel Hilty, and David Basin. Distributed Usage Control. *Communications of the ACM*, 49(9):39–44, September 2006.

[Rau04] Dominik Raub. Algebraische Spezifikation von Privacy Policies. Master's thesis, Universität Karlsruhe (TH), 2004.

[RS06] Dominik Raub and Rainer Steinwandt. An Algebra for Enterprise Privacy Policies Closed Under Composition and Conjunction. In *Proc. of Int. Conf. on Emerging Trends in Information and Communication Security (ETRICS)*, pages 132 – 146, 2006.

[SSA06] Stefan Sackmann, Jens Strücker, and Rafael Accorsi. Personalization in Privacy-Aware Highly Dynamic Systems. *Communications of the ACM*, 49(9):32–38, 2006.

[W3C06] W3C. Platform for Privacy Preferences (P3P) Project. `http://www.w3.org/P3P/`, 2006.

[WKK+07] Daniel Weiß, Jörn Kaack, Stefan Kirn, Maike Gilliot, Lutz Lowis, and Günter Müller at al. Die SIKOSA-Methodik: Unterstützung der industriellen Softwareproduktion durch methodische integrierte Softwareentwicklungsprozesse. *Wirtschaftsinformatik*, 49(3):188 – 198, 2007.

[WSP07] Web Services Policy 1.2 - Framework (WS-Policy). `http://www.w3.org/Submission/WS-Policy/`, 2007.

Ontology-enabled Documentation of Service-oriented Architectures with Ontobrowse Semantic Wiki

Hans-Jörg Happel[1], Stefan Seedorf[2], and Martin Schader[2]

[1] FZI Research Center for Information Technologies, Karlsruhe
happel@fzi.de
[2] University of Mannheim, Chair in Information Systems III
{seedorf|schader}@wifo3.uni-mannheim.de

Abstract: Documenting and maintaining an enterprise-wide service-oriented architecture (SOA) causes a substantial management effort, which should be addressed by intelligent, scalable solutions. A particular challenge is that business experts, developers, and software architects take different perspectives on a SOA, each favoring various description formats and sources, which leads towards a scattering of architectural information. Ontobrowse Semantic Wiki specifically addresses these issues by providing an ontology-based integration and documentation platform for architectural information. In this paper, we identify key issues arising from documentation and maintenance of a SOA by introducing the case of an insurance company. We give a detailed description of the Ontobrowse approach and its implementation, and explain how ontologies, artifact mappings, and architectural rules are created to support the Enterprise SOA case.

1 Introduction

The paradigm of service-oriented computing has lifted the development of business applications to a higher level of abstraction. Instead of thinking in technical categories like components or objects, software functionality is bundled in services, which correspond to business operations of the organization. Complex workflows can be realized by aggregating functionality from simple services. A concrete software infrastructure implementing this paradigm is called a service-oriented architecture (SOA) [HS05].

In an organization adopting a SOA, the standard working processes change for the multiple stakeholders involved, i.e., service developers, business experts, and software architects. First, service developers have to think in specification terms rather than taking an implementation view. Since services are black-box specifications, which hide the details of the internal realization, metadata describing their properties is crucial. In an enterprise-wide scenario, it therefore has to be documented which services are available, where the services are being deployed, how they can be invoked, and who is responsible for them.

Second, business experts are interested in available business functionality and operational efficiency. Since service-orientation leads to a rising level of alignment

between business processes and IT implementation, it is important to monitor and guide the development of the service landscape according to changing business requirements.

Third, software architects are interested in ensuring that services are specified and composed in such a way that key quality attributes such as performance, reusability, and modifiability are met. In order to achieve this, architectural patterns, rules, and policies are defined at the beginning of a SOA project. These are subsequently rolled out in service design decisions. However, as a SOA grows more complex, architects find it difficult to continously monitor if the current service definitions comply with the set of architectural rules.

Since all these perspectives on a SOA are tightly interwoven, it is desirable to provide an integrated view and access to different aspects of SOA documentation and maintenance, which allows the different stakeholders to incorporate change requests more rapidly and thoroughly and to better oversee consequences of their own actions. However, achieving this is challenging from both a technical and a functional point of view.

Technically, while a considerable number of standards to describe service properties like interface, behavior, and orchestration is available, this information is scattered in disjoint information spaces, which makes it hard for stakeholders to get an overview. Functionally, this is due to the fact that most SOA artifacts emerge “bottom-up” driven by the requirements of the respective tool suite of each stakeholder.

Thus, to align the different perspectives, a “top-down” layer is required, which serves as an integration and reference point for previously scattered information. In order to address the issues, we have developed Ontobrowse, a semantic wiki based on ontologies, which provides an infrastructure to extract knowledge from external sources [HS07]. In the case of an enterprise SOA, it can serve as single point of information, which allows to integrate different information from different stakeholder perspectives, thus covering both technical and business aspects of a SOA.

The remainder of the paper is structured as follows: In section 2, we present an Enterprise SOA case and derive the requirements for our documentation tool. We also address the shortcomings of existing approaches. In section 3, we shortly introduce the basic elements of our solution, namely ontologies and semantic wikis. In section 4, the architecture and realization of Ontobrowse is described. Section 5 demonstrates the setup with a concrete SOA ontology, artifact mapping an architectural rule to support the requirements. Finally, we summarize our findings in the conclusion.

2 Documenting and Maintaining an Enterprise SOA

In this section, we introduce the case of an enterprise SOA in an insurance company. It characterizes the systems, actors, and development artifacts in a concrete SOA environment. This helps us to identify the shortcomings of current SOA documentation practices and derive requirements for a suitable tool support.

2.1 Enterprise SOA Case

InsCorp Inc. is an insurance company, which has established itself as a top ten player for life insurance in its domestic market. The system landscape at InsCorp has continuously grown over the last decade and become highly heterogeneous. It consists of several legacy systems as well as databases, enterprise applications, ERP, and CRM systems. One major challenge is aligning the IT landscape to changing business requirements. In particular, cooperation with third-party vendors and product diversification has led to a high number of client systems with similar functionality. The current enterprise architecture makes managing change more and more difficult. In order to allow for further growth, InsCorp has commissioned ITCorp Inc. with a new project. The purpose is to develop a new enterprise-wide SOA, which meets the needs of its agile business environment.

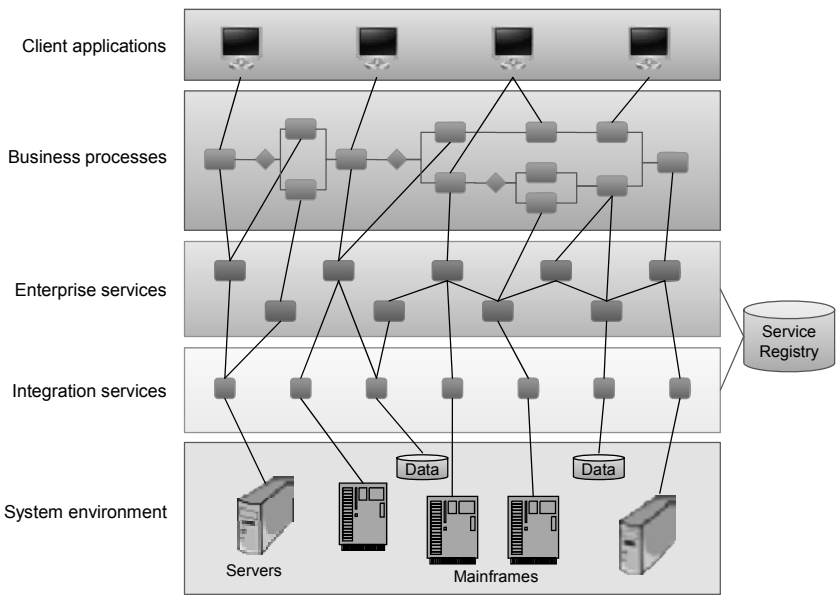

Figure 1: Organization of services in architectural layers

The development and evolution of a new architecture involves a number of different stakeholders who collaborate throughout the SOA lifecycle. The service architecture is organized in several logical layers as depicted in Figure 1. Different tasks and roles can be assigned to each layer. The *system environment* layer is maintained by several groups in the IT department. In the *integration services* layer, the functionalities of individual systems are exposed as Web Services to achieve technology abstraction. Maintainers of legacy systems and SOA architects have to cooperate in order keep these two layers in sync. The *enterprise services* layer aggregates basic services from the integration layer to support business activities. Business experts, process engineers, and software architects have to work seamlessly together to specify enterprise services and business objects. In the *business process* layer, process engineers realize workflows, which are

composed from the enterprise layer. Finally, application frontends in the *client application* layer execute these business processes to perform a business task. This top layer drives the evolution of the SOA, since the frontend applications require different business processes and data formats. Changing end user requirements can thus require modifications in all further layers.

Such modifications in turn result in considerable coordination overhead, since most of the layers depicted in Figure 1 are maintained by different stakeholders who stem from different organizations or at least different departments inside InsCorp.

These stakeholders document their views using different notations and formats. At InsCorp business analysts, responsible for *client applications*, use very generic tools such as Word and Excel for the functional description of services. In contrast, process engineers and service developers, responsible for the *business processes* and *enterprise services* layer, primarily work with technical specifications. Services are described in WSDL, the business objects are defined in XML Schema. Process engineers use a visual editor, which generates executable business processes in BPEL. Runtime information is captured using a custom-made service registry. At InsCorp, these tasks are completely managed by employees from ITCorp. Inc. The *integration services* and system level is partly maintained by technical departments of InsCorp., while a number of legacy systems is managed by different external contractors with specialized expertise.

Because various stakeholders contribute to the enterprise architecture, new challenges concerning architecture documentation and maintenance arise. Various issues stemming from this complex dependencies are reported by InsCorp. First, propagating changes across different layers involves considerable coordination effort and time. Different tools and data formats are used by various stakeholders, which requires many hand-crafted transformations. This is a frequent source of problems, especially if staff members vary, since people working at adjacent layers do not maintain any shared conceptualization of their collaboration beyond the technical artifacts and descriptions they exchange.

Furthermore, SOA artifacts are maintained in separate information spaces, which makes it difficult for other stakeholders to find relevant information for a task at hand. Also, much information gets lost at the interfaces between different layers, which leads to communication gaps– e.g., between business experts and developers– when there is no representation for the mapping between functional and technical service descriptions. While for example the response times of certain legacy systems might be important for a business analyst to cast a decision about some functionality, there is no place in the current setting, where such information could be stored.

From the perspective of InsCorp, this is highly problematic, since it creates a high dependency on the staff and work performed by ITCorp. Furthermore, it is neither possible to easily check the consistency of the overall SOA architecture, nor to estimate modification efforts in dependent layers, if changes are necessary. This inhibits InsCorp's ability to innovate and quickly react on market demands.

Although the original purpose of the SOA project was to reduce complexity and foster reuse, the large number of services and business processes makes it difficult to get an

overview of the architecture. Therefore, InsCorp and ITCorp look for a documentation tool, which integrates the various architectural views and which can be customized to project-specific needs.

2.2 Requirements

At first glance, the introduction of an enterprise-wide SOA simplifies the task of architecture documentation and maintenance due to a higher degree of standardization. However, as multiple stakeholders are involved a SOA also leads to a scattering of architectural information. Thus, a documentation tool should fit for different stakeholder viewpoints by creating an integration space.

One possibility for that is to go for a heavyweight, integrated SOA management suite, as it is offered by several vendors. However, these suites typically require complete migration of existing solutions and enforcement of a strict process across the whole architecture. For InsCorp, this is hardly possible, due to the complex organizational and technical ecosystem, which is already in place. As these stakeholders use dedicated formats and tools to describe their viewpoint a SOA documentation tool should be able to integrate various types of architectural information from existing environments, e.g., service specifications and their functional descriptions.

Considering this situation, we propose a lightweight, non-invasive solution that integrates as seamlessly as possible with the methods and tools already in use. Such a tool should neither replace nor interfere with existing tools and workflows, but provide a complementary, cohesive point of reference for all stakeholders participating in the development and maintenance of the SOA.

From the case in the previous section three main functional requirements for such a tool can be identified. These will be discussed in more detail below (c.f. Figure 2):

- Searching and browsing of SOA elements
- Checking the consistency of SOA elements
- Text documentation of SOA elements

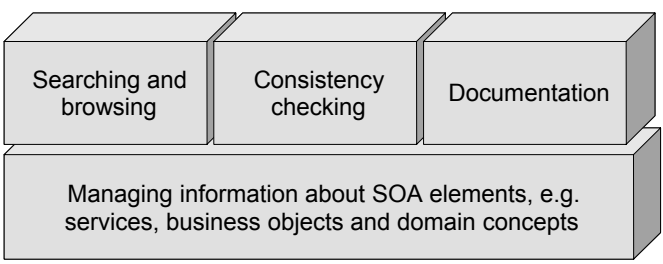

Figure 2: Overview of requirements

Searching and browsing of SOA elements is required to allow a better overview of the SOA for all stakeholders, e.g., architects, developers, and business experts. By describing the precise relations between services, business objects, and domain concepts, it is possible to gain a quick overview, e.g., which services require or return a certain business object. This enables an easy access to explore the list of available services, which may also promote the reuse of existing services.

Another requirement is consistency checking of SOA elements. What is meant here is not the syntactic consistency of the SOA, which is checked by the development-level tools, but consistency on a semantic level. Those are typically expressed as architectural and design rules, e.g., “Services may not call other services more than one layer apart” or “Services should not have more than five operations”.

Finally, the approach should enhance service documentation. Normally, there are two kinds of service documentation: technical descriptions, which are maintained by developers and business-oriented documentation residing in separate documents. This situation makes it difficult to get the complete information about a service, since documentation is distributed across physical storage locations and media types. The purpose is to create a single point of information for every service and business object. These elements of a SOA should be accessible by specifying a URL. In addition to accessing information, it should also be possible to add text documentation directly to a SOA element.

Besides that, the system has to support the management of SOA elements, such as services, business objects, domain concepts etc. Since the Enterprise SOA case requires a flexible solution, which is adaptable to the environment of a particular project, it must be possible to import arbitrary architectural descriptions that are not maintained internally but externally using project-specific formats and tools. Further, we need administrative features for governing this process such as updating and deleting information that has been acquired from external sources.

2.3 Related Work

Documenting and maintaining complex architectures has been a problem in Software Engineering for a long time. But as we will see in the following, existing work does not fulfill all requirements, which we described in the previous section. While there are systems and approaches, which support either searching and browsing, consistency checking or documentation for particular application scenarios, it will turn out that they lack flexibility to deal with both structured and unstructured information as it is required in a setting with diverse stakeholders.

Along with the increasing significance of SOA, new enterprise architecture tools have been emerging. These are usually geared towards managing the entire service life cycle, from analysis and design to versioning, deployment and monitoring. One shortcoming of these tools is that they lack openness since they are based on a relatively fixed metamodel. This may lead to problems when an organization deviates from a standard scheme, e.g., using other formats than WSDL or BPEL. In contrast, our solution is geared towards allowing maximum flexibility but at the same time enabling strong semantics. Since the stakeholders first have to agree upon one or more ontologies, the upfront investment setting up an environment can be higher. However, the costs can be reduced if existing ontologies and format mappings are reused.

As a SOA is only one specific instance of a software architecture, there are also more general approaches for enterprise architecture documentation. In most cases an enterprise architecture is represented using a number of different views (see [Kr95] for example). Depending on the particular view, formal or informal notations such as architecture description language (ADL) are used [MT00]. However, the case in section 2.1 substantially differs from the purpose of ADLs. Whereas ADLs focus on the specification and verification of a single view, we strive towards an integrated approach with a first-class representation for integrating all local views. Moreover, we want to include both formal and informal descriptions in this representation,

Universal modeling languages such as the Unified Modeling Language (UML) are also useful to describe a number of architectural views. Although the UML can be extended to cover various aspects, it does not include mechanisms for information integration and automated reasoning. The UML is also not useful for text documentation, which limits its applicability as the only representation language in the Enterprise SOA case.

Some researchers have proposed wikis for architecture and software documentation. Aguiar and David present a wiki-based approach to integrate heterogeneous software specification resources into a single document [AD05], while Bachmann and Merson investigate the advantages of wikis compared to other architecture documentation tools [BM05]. However, the approaches lack a formal model – information is managed in an unstructured way.

Moreover, formal ontologies have been proposed for architectural documentation [WF99] and for building "software information systems", describing the interrelationships of domain models and source code [We03]. These approaches are usually bound to a specific tool, which cannot be tailored to individual project needs or follow a very strict philosophy of software architecture.

To summarize, existing solutions for architectural documentation either impose a very strict formal model, which lacks the flexibility of documentation required in our case, or completely lack any formal model, which makes consistency checking and searching difficult. What is sought is an approach that allows to bridge varying degrees of structure.

Knowledge engineering, as a domain with similar problems, has recently bred the concept of "Semantic Wikis", which combine Wikis and ontologies [Vö06]. Since first applications in the domain of Software Engineering appear promising (see e.g. [De05]), we will investigate its suitability with respect to our requirements in the following section.

3 Foundations of the Ontobrowse Approach

In this section, we introduce the basic concepts of our solution. Revisiting our requirements in section 2.2, we regard ontologies and semantic wikis to be perfectly suited for our approach. Our goal is to model the structure of existing data (e.g., services defined in a WSDL file) and align these models to a top-level SOA ontology. At the runtime of our system, facts from existing artifacts can thus be automatically extracted and imported into our knowledge base. Modeling the information of the SOA domain and related development artifacts and processes in a SOA ontology will have the following benefits:

- Ontologies enable the integration of different aspects and data sources of the domain in question. By aligning the individual data source to a top-level ontology, the complete information can be searched and browsed in a unified way. Semantic links between different concepts (e.g., a process defined in a WSDL file and a user defined in an issue tracking system) can be modeled. Previously disjoint information such as e.g., the operations of a service and information about runtime behavior can thus be aggregated and presented in a single place (see e.g. Figure 7).

- Further, this integrated model of the SOA domain can be automatically checked for consistency. This can involve basic consistency checks, such as cardinality constraints (e.g., "a process should only have one owner") or more complex checks, which can be formulated using rule-based approaches (cf. section 5).

Finally, the ontologies and a knowledge base serve as backbone for a semantic wiki, where each entity in the knowledge base can be browsed, queried, documented, and semantically referenced.

3.1 Ontologies

An ontology in information systems (IS) provides the structure for a commonly agreed understanding of a problem domain. According to a widespread definition, "an ontology is an explicit specification of a conceptualization" [Gr93]. Ontology specifications have varying degrees of formalization; for example a shared vocabulary of terms can be regarded as lightweight ontology. Usually, ontologies are specified in a knowledge representation language using sets of concepts, relations, and axioms.

Ontologies qualify for the Enterprise SOA scenario because they serve as a unifying framework for different viewpoints and aim at reducing conceptual and terminological

confusion [UG96]. First, different stakeholders have to gain an enterprise-wide understanding of the problem domain. In our case, it includes an abstract model (conceptualization) of the concepts and relations together with their intended meaning, e.g., the meaning of "service" and "business object". Second, ontologies allow for the integration of heterogeneous information from various sources [St01]. This way, formalized knowledge can be more easily transferred and translated into different perspectives. Third, ontologies can be expressed using a knowledge representation language, which has a sound formal basis. This enables inference services and automated consistency checks, which is another requirement of our system. Recently, standards for ontology representation such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL) have emerged. RDF is a simple graph-like format for describing metadata about resources[1]. OWL[2] is defined on top of RDF(S) and provides a standard ontology vocabulary for describing ontologies based on description logics.

Due to these advantages ontologies have become the leveraging element in many knowledge management approaches [Ma03, OL98]. With the emerging vision of the Semantic Web [BHL01], ontologies have also attained increasing attention in the Software Engineering community. The potential applications in Software Engineering are manifold since ontologies can be used at development-time as well as runtime [HS06]. Our approach can be classified as ontology-enabled (cf. [HS06]) because it uses ontologies as its infrastructure for supporting development activities.

3.2 Semantic Wikis

Due to their low entry barriers and collaborative features, wikis are a lightweight approach to web-based content management, which allows multiple users to create documents on a shared subject of interest [LC01]. They have thus become a popular documentation tool in software processes (see [De05] for an overview). However, traditional wikis exhibit weaknesses when it comes to structuring the content of a wiki page. Although the set of "pages" forms a top-level structure, the underlying page content cannot be structured.

This has led to the idea of "semantic wikis". If some wiki content was structured and made machine-interpretable, a site like the Wikipedia could heavily benefit because its pages contain a lot of useful and potentially machine-processible knowledge [Vö06]. Several projects have thus proposed semantic extensions to the wiki approach. They all have in common that they allow structured knowledge to be described in a formal language, instead of processing solely hypermedia-based content. This is either done by appending metadata to wiki pages or by including knowledge inside the unstructured text by using extensions to the wiki markup language. The latter approach is used by the SemanticMediaWiki project [Vö06], which extends the existing wiki markup to enrich hyperlinks between wiki pages with semantic relations.

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/owl-features/

Semantic wikis interpret wiki pages as entities, and hyperlinks between wiki pages as relations among entities. Due to the additional semantic descriptions the implicit structure is made explicit, and a machine-processible knowledge model can be derived. Current semantic wikis differ concerning the creation and maintenance of this explicit kind of knowledge. While some require upfront knowledge engineering efforts, other approaches allow for continuous refinement and formalization of knowledge by the users. As we will line out in the following section, we prefer a dual approach.

Clearly, semantic wikis are a prime candidate for knowledge sharing in our case, because they provide a user-friendly way for searching and browsing structured information. Another advantage is that they combine informal with formal descriptions, thus closing the gap between the business-oriented and technical perspective on an architecture.

4 Ontobrowse Semantic Wiki

Based on wikis and ontologies as building blocks, we now describe the general architecture and prototypical implementation of our approach.

4.1 General Architecture

In the previous sections we identified the integration of architectural information from external sources as one key requirement. In Ontobrowse we thus distinguish two layers. The first layer is the artifact layer where the stakeholders maintain descriptions of services using their own tools and formats. In order to integrate the various artifacts into a mutual knowledge base, they have then to be mapped according to an ontology. Moreover, we need the semantic wiki as a user interface for presenting, querying and searching the knowledge base.

The resulting conceptual architecture of Ontobrowse is depicted in Figure 3. The integration layer includes the following components: a Web interface, a wiki manager, an ontology API to access the knowledge base, and a plugin manager.

The central part in the integration layer is the knowledge base, which is formed by one or more ontologies and instance data. It is processed using an ontology API and an underlying reasoner. While the ontologies define the knowledge structure, i.e., the boundaries in which instances can be described, instance data refers to the individual objects and their property descriptions conforming to the ontology. For example, a SOA ontology may specify the concepts “service” and “business object” together with their properties and axioms. The instances are represented by actual services and business objects developed in a SOA project. Each concept, relation, or individual is displayed by the wiki manager as a “wiki page”. It contains properties that make statements about this page, e.g., a business object which is semantically described by a domain concept. We also refer to a wiki page as an “entity”, because it is contained in the knowledge base and can be requested with a unique identifier (URI).

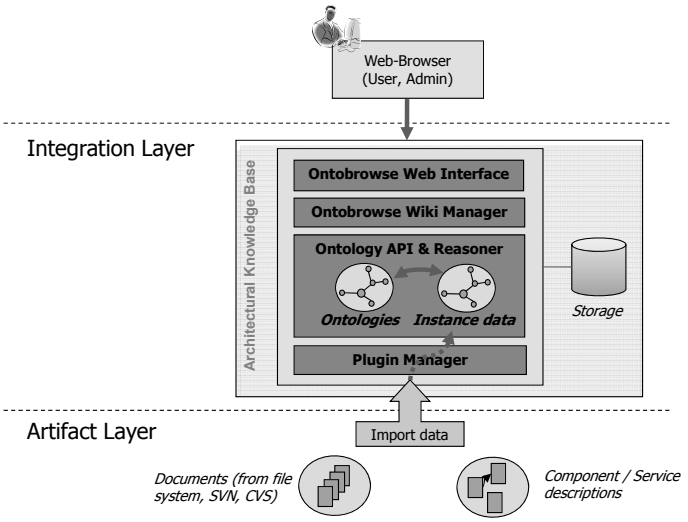

Figure 3: Ontobrowse architecture [HS07]

The wiki manager bundles the functions for fulfilling the requirements, such as processing page requests, editing textual documentation and instance property values, searching and deductive querying, and user authentication. Entity (page) descriptions are returned by an ontology API, which wraps the underlying reasoner and ontology processing tools.

Typically, ontologies are constructed upfront using an ontology editor such as Protégé[3] and uploaded by an administrator using the wiki manager. Within the knowledge structure defined by ontologies, it is possible to add instance data in two different ways: First, a wiki user can use the interface to describe properties – may it be text-based or metadata-based – about instances of concepts. Second, external tools can plug into the wiki application and map architectural description resources to instances in the knowledge base.

This leads us to the plugin manager. To a great extent, the instance data is embodied in applications and artifacts, which are managed outside the wiki, e.g., service specifications in the Enterprise SOA case. This data has to be imported from external sources, such as configuration management systems. The plugin manager allows mapping external artifacts to instance data and add this data to the knowledge base. This component exposes standard interfaces, which allow tools to retrieve artifacts, map them according to an ontology, and create or update instance data in the knowledge base. As an example, the operations of a service can be automatically extracted from service description artifacts.

[3] http://protege.stanford.edu

4.2 Design and Implementation

In this section, we will describe a concrete implementation of the conceptual architecture according to the layers of the architecture, which were introduced in the previous section.

Since SOA artifacts tend to be maintained in various heterogeneous systems and data sources, such as XML descriptions, documents, or databases, the artifact layer deals with the extraction of facts from these sources,. As a framework for this task, we use the Open Source framework Aperture[4]. Aperture comes with a number of physical connectors, such as for crawling file systems or the web, which we complemented by crawlers for SVN and CVS repositories, as well as direct connections to JDBC databases and issues tracking systems such as JIRA.

In Aperture, the objects that are crawled from those sources are directed to so-called Extractors. These extract metadata in RDF. Besides the built-in extractors for various common document formats (e.g., Microsoft Office and PDF), we added extractors for WSDL, JIRA, and Java source code.

The artifact layer is extensible in two ways: additional repositories of supported types (such as an additional SVN repository to crawl) can be easily configured by XML files. To include new data sources, Aperture provides a modular infrastructure, which just requires the implementation of two Java interfaces.

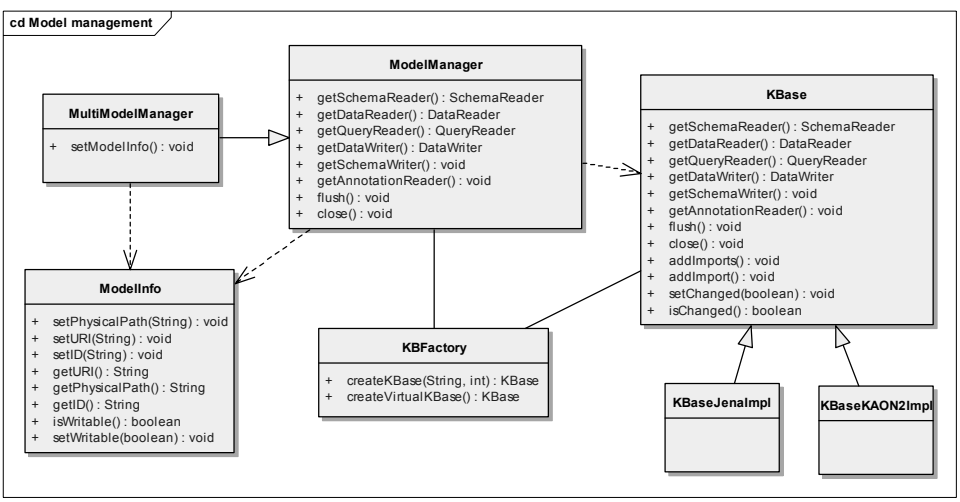

Figure 4: Ontobrowse Model Manager

The metadata extracted by the Aperture crawlers is written to the Ontobrowse knowledge base. In order to encapsulate the concrete metadata store used, we developed an abstraction API called "KOntoR API", which we briefly describe in the following.

[4] http:// aperture.sourceforge.net/

The API consists of two major parts: the model management package and the ontology API. The model management mainly consists of the ModelManager and ModelInfo classes (cf. Figure 4). ModelInfo encapsulates a file serialization of an ontology. It serves as a parameter for ModelManager for dealing with only one ontology or MultiModelManager, when dealing with a set of ontologies.

The ModelManager uses a KBFactory class to instantiate a KBase using either a KAON2 reasoner[5] or a Jena metadata store[6] as a backend. However, the concrete backend remains hidden to the using classes. After instantiation, the content of the ontologies can be accessed and modified using the different interfaces provided by ModelManager.

There are interfaces for reading certain ontology information (e.g., SchemaReader, DataReader), writing data (e.g., DataWriter), and querying (QueryReader). These interfaces again have special implementations for each backend (e.g., for KAON2 or Jena). The methods of the interfaces map to the atomic entities of the data API, which will be described in the following.

The Ontology API serves as a lightweight, partial representation of a graph structure. The subsystem consists of the following classes (see Figure 5):

KBEntity: This is the abstract base class of an entity in the knowledge base. An entity is characterized by a name (label) and a URI.

Concept: This is the representation of a concept (or "class" in OWL terms) in a knowledge base. It is a lightweight representation, since it does not include information about individuals, datatypes, or object properties.

RichConcept: This is a heavyweight representation of a concept. In contrast to the Concept class, it contains information about the datatypes and the properties.

Individual: This is a lightweight representation of an individual (also called "instance") in a knowledge base.

RichInvidual: This is a heavyweight representation of an individual. It contains datatype and object properties with values. Note that values of object properties are again (lightweight) Individuals, which can be resolved to RichIndividuals.

ObjectProperty/RichObjectProperty: The ObjectProperty classes encapsulate relations between concepts, which exist in the knowledge base.

DatatypeProperty/RichDatatypeProperty: The DatatypeProperty classes represent attributes of concepts, which are of base types such as String, numbers, or date values.

[5] http://kaon2.semanticweb.org
[6] http://jena.sourceforge.net/

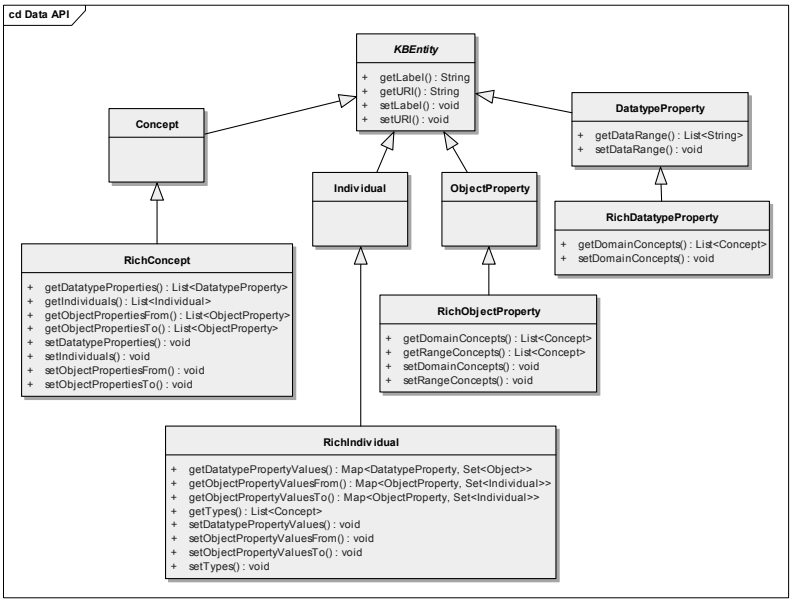

Figure 5: Ontobrowse Ontology API

Existing Java APIs for OWL knowledge bases (e.g., Protégé API[7], OWL-API[8]) are mostly based on (a) representing the whole ontology graph in memory and (b) supporting the full set of axioms for the underlying knowledge representation language.

In contrast to this, our design goals are:

- Provide a lightweight and stateless API for knowledge base access
- Focus on instance retrieval and manipulation, and omit sophisticated schema manipulation
- Provide an abstraction layer for ontology stores/engines

Thus, our API does not replace, but complements the APIs of existing ontology stores. We rely on the existing implementation e.g., for loading ontologies and executing reasoning tasks at the low level, and provide a high level representation, which abstracts from most complexities in ontology handling. Due to the abstraction layer, a further advantage of our API is that it abstracts from the specifics a concrete knowledge representation language. Our API is not necessarily limited to Semantic Web languages, since it could also have an implementation based on relational databases.

Currently, we have implementations of our API for the Jena Semantic Web framework, as well as for the KAON2 OWL reasoner.

[7] http://protege.stanford.edu/plugins/owl/api/
[8] http://owl.man.ac.uk/api.shtml

Besides the ontology API, Ontobrowse offers service interfaces for importing and updating ontologies and for managing wiki pages as well as user accounts.

While the backend services can be accessed via arbitrary clients, our standard user interface is a web application implemented with Java Server Faces[9]. Currently, this web application includes dialogs to browse the knowledge base (list concepts, view concepts and instances– see e.g., Figure 5, specify and execute SPARQL queries and for user management. We are currently extending the user interface to allow for editing property values and to add relations to link different entities in the knowledge base.

5 Setting Up Ontobrowse in a SOA Environment

In this part, we describe the necessary steps for setting up Ontobrowse in a concrete SOA environment. Initially, all stakeholders need to agree upon a shared conceptual structure, a so-called "SOA ontology" (cf. Figure 6). This ontology should capture a shared understanding of both business experts and technical people. Typically, it includes concepts like "service", "interface", "business object", and "domain concept". On the one hand, "service" defines data types and properties from the technical domain, such as "version" and "hasInterface". On the other hand, it includes properties relevant in the business view, such as "refersToDomainConcept" to reference a project glossary term. The specification has to be carried out by ontology engineers, creating an ontology file with an editor such as Protégé. An ontology file is uploaded to Ontobrowse via the Web interface and subsequently processed by the Ontology API.

[9] http://java.sun.com/javaee/javaserverfaces/

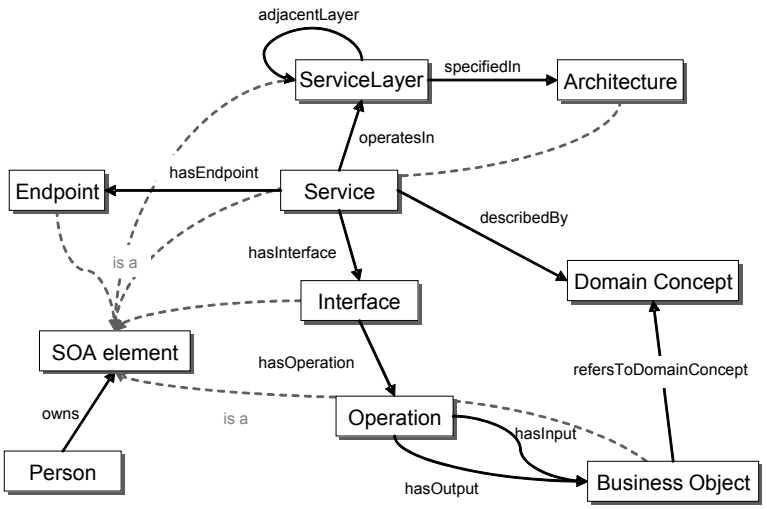

Figure 6: SOA ontology

The stakeholders can also decide whether they reuse existing ontologies. Potentially useful sources include the foundational ontologies being developed within Semantic Web Services [Ak05, ES05, OW04] and the Web Services Architecture [W3C04]. Of course, it is possible to develop several modular ontologies covering various information needs, e.g., project management and organizational structure. Instance data corresponding to the SOA ontology may be created either directly in the wiki or imported from external sources by defining plugins. This ensures high flexibility and enables to augment SOA elements with additional descriptions.

Plugins perform the actual mapping of instances from an external source into the knowledge base, e.g., from WSDL service descriptions maintained in a file system to service descriptions in the wiki. Here, we give an example how the mapping works for WSDL. However, the process is analogous for other formats, e.g., the Business Process Execution Language for service composition.

First, a one-way mapping between WSDL service descriptions and the SOA ontology is defined. In order to accommodate service properties such as version and architectural layer, we extended the WSDL format. The actual mapping is executed by a Java program, which conforms to the Ontobrowse plugin interface. It takes a WSDL file as input and performs a set of actions for adding instances, properties and attributes to the knowledge base with the Ontology API. A wiki administrator configures the input sources (CVS, file system) and update types (manual, timer task, update event). Based on this configuration the plugin manager component is responsible for updating the knowledge base automatically.

Figure 7: Concept page for "service"

Once the initial structure and wiki content has been created, it is possible to access the knowledge base through the Web interface. First, users can quickly gain an overview by starting with a concept page. For example, the page for the concept "service" shows the sub-concepts, all instances to that concept as well as the incoming and outgoing properties (cf. Figure 7). A user can then navigate to a service to read its detailed description. Second, there is a full text search of all entities in the knowledge base. Third, there is also the possibility for looking for very specific knowledge. A query interface enables users to define chained queries consisting of sentences with *subject*, *predicate* and *object* (e.g., all services "x" defining interface operations with the output "Customer"). Matching entities are returned for the variables defined by the query.

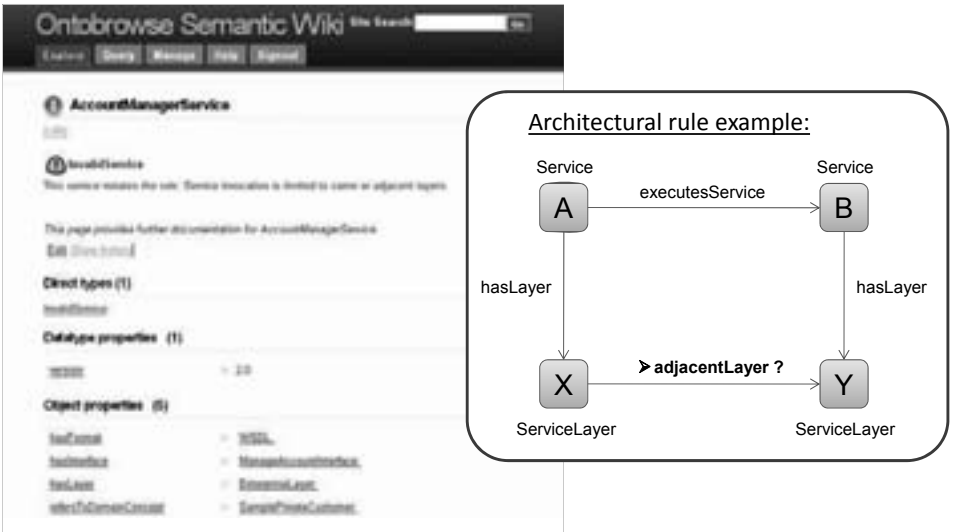

Figure 8: A service violating an architectural rule

Finally, the SOA ontology can be enhanced by rules, which enable automatic consistency checking of entities and generation of new knowledge. So far, we have included experimental support for DL-safe SWRL[10] rules in the KAON2 configuration. One application scenario is the formal definition of architectural rules, which are usually only informally documented by software architects. The semantic wiki makes the violation of these rules explicit, thus supporting their enterprise-wide enforcement. For example, we stated the rule "services may not call other services more than one layer apart" (see section 2.2) as visualized in Figure 8:

$$\text{executesService}(A, B) \wedge \text{hasLayer}(A, X) \wedge \text{hasLayer}(B,Y) \wedge \neg\, \text{adjacentLayer}(X,Y) \wedge \neg\, \text{owl:sameAs}(X,Y) \Rightarrow \text{InvalidService}(A)$$

Any entity violating this rule is categorized as an "invalid service". By using ontology annotations for the concept "invalid service" the Web interface can display a warning to the user. Alternatively, it is possible to filter for all invalid services using the query interface. To this end, Ontobrowse not only improves the navigation, documentation, querying and searching but also contributes to the quality of an Enterprise SOA.

6 Conclusion

In this paper, we described an approach based on ontologies and semantic wikis, which tackles key issues in the documentation of an Enterprise SOA. The SOA case revealed the distributed character of the SOA development process, which has been insufficiently addressed so far. Because an Enterprise SOA not only involves multiple roles, but also

[10] http://www.w3.org/Submission/SWRL/

brings different organizational units and external service providers together, the responsibilities (and with it architectural information) are inherently distributed.

Although a SOA leads to a higher degree of standardization at first glance, it nevertheless involves different views, which are either technical or business-oriented. This results in a high number of heterogeneous, locally maintained SOA artifacts with varying degrees of formalization. What is sought after is therefore both a “common language” shared by all stakeholders and a first-class representation for different types of architectural information. As pointed out in this paper, ontologies can provide the means for solving the terminological and the information integration problem. Semantic wikis on the other hand, provide a flexible way for accessing this information, e.g. browsing searching, and semantic querying. Most importantly, the proposed solution can be tailored to project-specific needs by defining one or more ontologies to set up the initial structure of the wiki.

References

[AD05] Aguiar, A.; David, G.: WikiWiki weaving heterogeneous software artifact. In: Proc. of the 2005 international symposium on Wikis, San Diego, CA, 2005, pp. 67-74.

[Ak05] Akkiraju, R.; et al.: Web Service Semantics - WSDL-S. W3C Member Submission, 2005.

[BM05] Bachmann F.; Merson, P.: Experience Using the Web-Based Tool Wiki for Architecture Documentation. Technical Note CMU/SEI-2005-TN-041. September 2005.

[BHL01] Berners-Lee, T.; Hendler J.; Lassila, O.: The Semantic Web. Scientific American. May, 2001.

[De05] Decker, B. et.al.: Self-organized Reuse of Software Engineering Knowledge supported by Semantic Wikis. In: Proc. of Workshop on Semantic Web Enabled Software Engineering, Nov. 2005.

[ES05] ESSI WSMO: Web Service Modeling Ontology (WSMO). http://www.wsmo.org/, 2005.

[Gr93] Gruber T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5, 1993, pp. 199-220.

[HS06] Happel, H.-J.; Seedorf, S.: Applications of Ontologies in Software Engineering. In: Proc. of Workshop on Sematic Web Enabled Software Engineering" (SWESE) on the ISWC 2006, Athens, Georgia, November 5-9, 2006.

[HS07] Happel, H.-J.; Seedorf, S.: Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures. In: Proc. of the 19th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Boston, USA, July 9-11, 2007, pp. 506-512.

[HS05] Huhns, M.H.; Singh, M.P.: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, vol. 9, no. 1, 2005, pp. 75-81.

[Kr95] Kruchten, P.: The 4+1 View Model of Architecture. In: IEEE Softw. 12 November, Nr. 6, 1995, pp. 42-50.

[LC01] Leuf, B., Cunningham, W.: The wiki way: Quick collaboration on the web. Addison-Wesley, 2001.

[Ma03] Maedche, A. et.al.: Ontologies for En-terprise Knowledge Management. IEEE Intelligent Systems ,18, 2003, pp. 26-33.

[MT00] Medvidovic, N.; Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. In: IEEE Trans. Software Eng. 26(1): 2000, pp. 70-93.

[OL98] O'Leary; D.E.: Using AI in Knowledge Management: Knowledge Bases and Ontologies. IEEE Intelligent Systems, 13, 1998, pp. 34-39.
[OW04] OWL Services Coalition: OWL-S Semantic Markup for Web Services. 2004.
[St01] Staab, S. et.al.: Knowledge Processes and Ontologies. IEEE Intelligent Systems, 16, 2001, pp. 26-34.
[UG96] Uschold, M.; Gruninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review, 11, 1996, pp. 93-155.
[Vö06] Völkel, M. et.al.: Semantic Wikipedia. In: Proceedings of the 15th International Conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006.
[W3C04] W3C: Web Services Architecture. W3C Working Group Note, 11 February, 2004.
[We03] Welty, C.A.: Software Engineering. In: Description Logic Handbook, 2003, pp. 373-387.
[WF99] Welty, C.A.; Ferrucci D.A.: A Formal Ontology for Re-Use of Software Architecture Documents. ASE, 1999, pp. 259-262.

Architectural Principles and Components of Adaptive Process Management Technology

Manfred Reichert[1], Peter Dadam[1], Stefanie Rinderle-Ma[1],
Martin Jurisch[2], Ulrich Kreher[2], Kevin Göser[2]

[1]Institute of Databases and Information Systems, Ulm University, Germany
{peter.dadam, manfred.reichert, stefanie.rinderle}@uni-ulm.de
[2]AristaFlow GmbH, Germany
{martin.jurisch, ulrich.kreher, kevin.goeser}@aristaflow.com

Abstract. Process-aware information systems (PAIS) must not freeze business processes, but should enable authorized users to deviate from the implemented workflows on-the-fly and to dynamically evolve them over time. While there has been a lot of work on the theoretical foundations of dynamic process changes, there is still a lack of implemented PAIS providing this dynamics. Designing the architecture of such adaptive PAIS, however, constitutes a big challenge due to the high complexity coming with dynamic changes. Besides this, performance, robustness, security and usability of the PAIS must not be affected by the added flexibility. In the AristaFlow project we follow a holistic approach to master this complexity. Based on a conceptual framework for adaptive process management, we have designed a sophisticated architecture for next generation process management technology. This paper discusses major design goals and basic architectural principles, gives insights into selected system components, and shows how change support features can be realized in an integrated and efficient manner.

1 Introduction

In today's dynamic business world enterprises must be able to quickly and flexibly react to changes in their environment [Guen06,RRD03a,RRD04a,WRR07]. Therefore, companies have recognized business agility as a competitive advantage, which is fundamental for being able to cope with trends like increasing product and service variability, faster time-to-market, and business-on-demand. *Process-aware information systems* (PAIS) offer promising perspectives in this respect and a growing interest in aligning information systems in a process-oriented way can be observed [BDR03,LeRe07, Müll06,Wesk07]. As opposed to data- or function-centered information systems, PAIS are characterized by a strict separation of process logic and application code. In particular, most PAIS describe process logic explicitly in terms of a *process template* providing the *schema* for process enactment. Usually, the core of the process layer is built by a *process management system*, which provides generic functions for modeling, executing, and monitoring processes [Wesk07]. This allows for a separation of concerns, which is a well established principle for increasing maintainability and for reducing cost of change; i.e., changes to one layer can be performed without affecting other layers.

The ability to deal with process change has been identified as one of the most critical success factors for PAIS [LRW08,RDB03,ReDa98,RRD04a,WRR07]. Through the described separation of concerns PAIS facilitate changes significantly. However, enterprises are still reluctant to adapt PAIS once they are running properly. High complexity and cost of change are mentioned as major obstacles for not fully leveraging the potential of PAIS [MRB08]. To improve this situation more flexible PAIS are needed enabling companies to capture real-world processes adequately without leading to mismatches between computerized processes and those running in reality. Instead, users need to be able to deviate from the predefined processes if required and to evolve process implementations over time [RRD03a]. Such changes must be possible at a high level of abstraction and without affecting PAIS consistency and robustness [RRD04a-d].

Basically, process changes may take place at the *instance* or the *type level*. Process instance changes often have to be carried out in an ad-hoc manner to deal with exceptional situations [ReDa98,DRK00,LeRe07]. Such *ad-hoc changes* must not affect system robustness or lead to errors in the sequel. Process type changes, in turn, correspond to continuous changes of a process schema (also denoted as *process schema evolution*) to deal with evolving needs [RRD04a-d]. Regarding long-running processes, it might also require the migration of already running process instances to the new schema version. Important challenges in this context are to perform such instance migrations on-the-fly, to guarantee compliance of the migrated process instances with the new process schema, and to avoid performance penalties [Rind06,RRD04a-b].

The design of adaptive process management technology constitutes an enormous challenge. On the one hand such technology should allow for efficient process enactment as well as for enterprise application integration. On the one hand it has to provide support for dynamic process changes. When designing such a technology we have to cope with many trade-offs. For example, complexity of dynamic changes increases the higher expressiveness of the used process modeling language becomes. Further, complex interdependencies between the different features of adaptive PAIS exist that must be carefully understood to avoid implementation gaps. Process schema evolution, for example, requires integrated support for high-level change patterns, schema versioning, change logging, on-the-fly instance migrations, and dynamic worklist adaptations. Finally, even if the conceptual pillars of such a next generation process management technology are well understood, it will still be a quantum leap to implement advanced features in an integrated, efficient and robust manner.

In the AristaFlow project we have followed a holistic approach to tackle these challenges. Based on a sophisticated conceptual framework for dynamic process changes, which we developed in the ADEPT project [ReDa98,RRD04b], we have designed the architecture of the ADEPT2 process management system and prototypically implemented it. From the very beginning, one of the primary design goals has been process flexibility. This paper summarizes major architectural principles, gives insights into ADEPT2 system components and their interactions, and shows how change support features can be realized in an integrated and efficient manner within such architecture.

The remainder of this paper is organized as follows: Section 2 presents background

information needed for understanding this paper. Section 3 summarizes architectural principles applied during the design of the ADEPT2 system and its components. Section 4 shows how ad-hoc changes and schema evolution are enabled within this architecture. Section 5 discusses related work and Section 6 concludes with a summary.

2 Conceptual Framework for Dynamic Changes in ADEPT2

We developed a framework for dynamic process changes in previous projects [ReDa98, RRD04b]. In this paper we use it as conceptual pillar for designing the ADEPT2 system architecture. ADEPT2 covers changes at both the process instance and the process type level. This, in turn, allows for *ad-hoc flexibility* as well as for *process schema evolution*.

Ad-hoc flexibility. At the instance level the ADEPT2 framework allows for ad-hoc deviations from the pre-modeled process schema (e.g., to insert, delete, or move activities). Such ad-hoc changes do not lead to unstable system behavior, i.e., none of the guarantees (e.g., absence of deadlocks) achieved by formal model checks at buildtime are violated due to dynamic changes [ReDa98]. ADEPT2 provides a complete set of high-level change patterns for defining ad-hoc deviations; e.g., authorized users may dynamically add new activities or jump forward in the flow of control [RDB03, WRW05]. ADEPT2 defines pre- and post-conditions for all operations to ensure correctness. Further, all complexity associated with the adaptation of instance states, the re-mapping of activity inputs/ outputs, the handling of missing input data, or the problem of deadlocks is hidden to a large degree from users.

Process schema evolution. To cope with business process changes (e.g., due to reengineering efforts), ADEPT2 allows for quick and efficient adaptations of process templates (i.e., schema changes at type level) – in the following denoted as *process schema evolution* [RRD04b]. When updating a process template, usually, related process instances are finished according to the old schema version, while future instances are derived from the new one. However, such rigid approach is not adequate for long-running processes [LeRe07]. Here, the challenge is to *propagate* respective schema changes to already running instances of this process template as well; i.e., to migrate these process instances to the new schema version of the respective process template .

The on-the-fly migration of a collection of process instances to a modified process template must not violate correctness and consistency properties of these instances. Therefore, we need a general principle for arguing whether a process instance is compliant with an updated schema [RRD04a,RRD04b]. The ADEPT2 change framework uses a well-defined correctness criterion, which is independent of the underlying process meta model and which is based on a relaxed notion of trace equivalence. This compliance criterion considers control as well as data flow changes, ensures correctness of instances after migration, works correctly in connection with loop backs, and does not needlessly exclude instances from migrations. To enable efficient compliance checks, precise and easy to implement compliance conditions are defined for each change operation (see Fig. 1 for an example). Finally, ADEPT2 automatically adapts the states of compliant instances when migrating them to an updated schema.

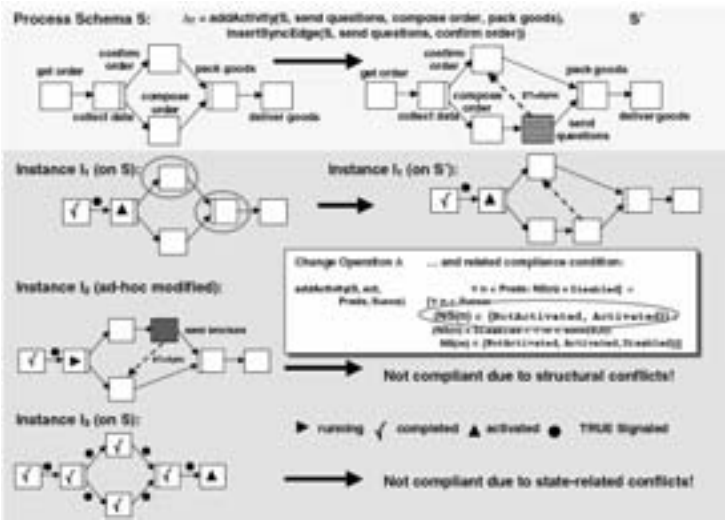

Figure 1: Process Schema Evolution (Conceptual View)

When designing ADEPT2 we have looked at the picture as a whole. In particular, we have not considered the different kinds of changes in an isolated manner, but have investigated their complex interdependencies as well. For example, the correct handling of concurrent process changes is crucial in order to cover all practical cases [RRD04c-d]. In this context, we have dealt with the question how to propagate process template changes to related process instances which are in different states and to which various ad-hoc modifications have been previously applied. For such *biased* instances, current instance schema differs from the schema of the original template. Therefore, change propagation must be accomplished while considering certain correctness constraints to avoid inconsistencies. In this context, ADEPT2 excludes state-related, structural, and semantical conflicts between concurrent changes [RRD03a,RRD04c,RRD04d].

As example consider Fig. 1 where a new template version S' is created from a process template S based on which three instances are running. Instance I_1 can be migrated to the new process template version. By contrast, instances I_2 and I_3 cannot migrate. I_3 has progressed too far and is therefore not compliant with the updated template schema. Though there is no state conflict regarding I_2 this instance can also not migrate to S'. I_2 has been individually modified by an ad-hoc change which is conflicting with the template change. More precisely, when propagating the process template change to I_2 a deadlock-causing cycle would occur. The ADEPT2 change framework provides efficient means to detect such structural conflicts [RRD03]. Basic to this are sophisticated conflict tests. In summary, we restrict propagation of a *process template change* to those instances for which the change does not conflict with instance state or previous ad-hoc changes. – So far, we have focused on our conceptual change framework, which constitutes the basis for the proper design of the ADEPT2 system architecture. Following sections illustrate how we have implemented this conceptual framework.

3 Design Principles and Components of the ADEPT2 Architecture

The design of the ADEPT2 system has been governed by a number of well-defined principles in order to realize a sustainable and modular system architecture. Considered design principles include general architectural aspects as well as conceptual issues concerning the different system features. Our overall goal is to enable ad-hoc flexibility and process schema evolution (cf. Section 2), together with other process support features, in an integrated way, while ensuring robustness, correctness, extensibility, performance and usability of the system at the same time. This section summarizes major design principles of the ADEPT2 architecture and gives an overview of its components.

3.1 General Design Principles and Goals

High-end process management technology has a complexity comparable to database systems. To master this complexity a proper and modular architecture is needed with clear separation of concerns and well-defined interfaces. This is fundamental to enable exchangeability of implementations, to foster extensibility of the architecture, and to realize autonomy and independency of system components to a large extent. To meet these goals the overall architecture should be layered. Thereby, components of lower layers must hide as much complexity as possible from upper layers. Basic components must be combinable in a flexible way to realize higher-level services like *ad-hoc flexibility* or *process schema evolution*. To achieve this, ADEPT2 components are reused in different context making use of sophisticated configuration facilities.

Process management systems should provide sophisticated buildtime and runtime components to the different user groups (e.g., process participants, process administrators, process designers). This includes buildtime tools for modeling, verifying and testing processes, runtime components for monitoring and dynamically adapting process instances, and worklist clients for accessing upcoming tasks. Many applications, however, require adapted user interfaces and functions to integrate process support features in the best possible way. On the one hand, provided user components should be configurable in a flexible manner. On the other hand, all functions offered by the process management system should be made available via application programming interfaces (APIs) as well. In particular, advanced system functions (e.g., ad-hoc changes or process schema evolution) should be accessible via such programming interfaces as well.

Implementation and maintenance of the different system components should be as easy as possible. Therefore each component is kept as simple as possible and only has access to the information needed for its proper functioning. Furthermore, communication details are hidden from component developers and independency from the used middleware components (e.g., database management systems) is realized.

To enable maintainability, extensibility and usability of the different system components we need a proper conceptual design for the ADEPT2 system architecture. We do not give an in-depth discussion of all considered design goals, but illustrate our main philosophy by means of two examples. Though these basic design principles apply to

modern software architectures in general, we explicitly list them here to make clear how they affect the design of adaptive process management technology:

- *Reuse of code fragments*: A major design goal for any complex system architecture is to avoid code redundancies. For example, components for process modeling, process schema evolution, and ad-hoc process changes are more or less based on the same set of change operations. This suggests to implement these operations by one separate system component, and to make this component configurable such that it can be reused in different context. Similar considerations have been made for other components (e.g., visualization, logging, versioning, and access control). This design principle does not only reduce code redundancies, but – as a consequence – results in better maintainability, decreased cost of change and reduced error rates.
- *Extensibility of system functions*. Generally, it must be possible to add new components to the overall architecture or to adapt existing ones. Ideally, such extensions or updates do not affect other components; i.e., their implementations must be robust with respect to changes of other components. As example assume that the set of supported change operations shall be extended (e.g., to provide higher level change patterns to users [WRR07]). This change, however, should neither affect the component realizing process schema evolution nor the one enabling ad-hoc flexibility. In ADEPT2 we achieve this by internally mapping high-level change operations to a stable set of low-level primitives (e.g., to add/delete nodes) [Rind06].

3.2 Overview of the ADEPT2 Architecture and its Components

Figure 2 depicts the overall architecture of the ADEPT2 process management system. Its development has been based on the design principles discussed before and on the experiences we gathered during implementation of ADEPT1 [RRD03b]. ADEPT2 features a layered and service-oriented architecture. Each layer comprises different components offering services to upper-layer components. The first layer is a thin abstraction on SQL, enabling a DBMS independent implementation of persistency. The second layer is responsible for storing and locking different entities of the process management system (e.g., process templates and process instances). The third layer encapsulates essential process support functions including process enactment and change management. The topmost layer provides different buildtime and runtime tools to users, including a process editor and a process monitoring component.

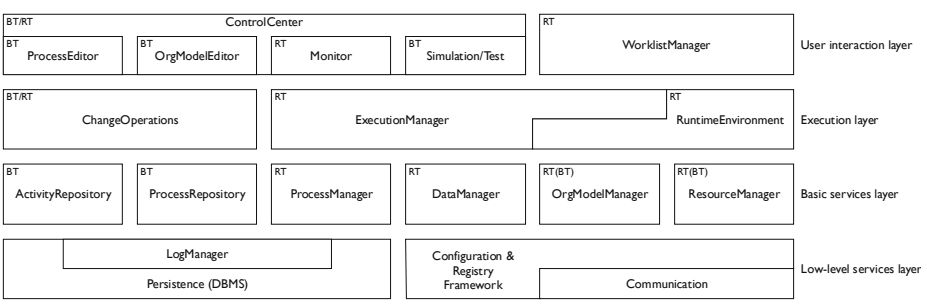

Figure 2: Basic Architecture of ADEPT2 (BT: Buildtime; RT: Runtime)

3.2.1 Layer with Low-level Services

This first layer of the ADEPT2 architecture comprises basic services which accomplish tasks like logging, persistency management, configuration support and communication. Idiosyncrasies of the used middleware services are hidden from upper-layer components. This allows us to use different database management systems or to exchange communication middleware without need for adapting implementations of upper layers.

Configuration & Registry Framework: This component provides the basic infrastructure for configuring and managing the different system components of the ADEPT2 architecture, and for enabling inter-component communication. The developed framework allows to start, manage and terminate ADEPT2 components (e.g., *ProcessManager*) as well as their services (e.g., managing instance data), and to flexibly configure them for use in different context. In addition, a generic interface is provided to realize communication between ADEPT2 components. Thereby, communication details (e.g., concerning transport protocols, interaction styles or message formats) are hidden from the components using this interface. For example, it remains transparent for these components whether the services they request are running locally or remotely.

LogManager: ADEPT2 logs all system events occurring at build- and runtime [Rind06, RJR07]. This includes events like changes in the state of a process instance, structural changes at type or instance level, or access to process data elements. The *LogManager* provides a generic interface based on which upper-layer components can log whatever events they want. Persistency is handled by a separate sub-component of the *LogManager*, which hides details of the underlying storage management component. This allows us to use different persistency managers (e.g., relational DBMS, XML files, flat files) without affecting implementation of upper layers.

3.2.2 Layer with Basic Services

Components of this layer provide basic services for managing build- and runtime data of the process management system and for making it available to upper-layer components.

***ActivityRepository*:** This system component manages the activity templates based on which processes can be composed and executed. An activity template encapsulates all information needed for working on a particular task. In particular, it connects the activity to an application component. Thereby, details of the used component model (e.g., Enterprise Java Beans, (D)COM or Web services) are hidden from other ADEPT2 system components. Activity templates comprise additional information needed for proper activity execution. Based on it, for example, one can figure out whether the associated application component can be interrupted or aborted during runtime.

***ProcessRepository*:** This component manages process templates and their meta data. Similar to activity templates, process templates can be used as building blocks when composing a new process model. Note that this allows for the realization of sub processes in an easy and intuitive manner. Furthermore, the *ProcessRepository* component manages all versions of a process template and the information needed to derive them from each other; i.e. change logs [RJR07] regarding the process templates.

***ProcessManager*:** While the above components manage buildtime data, the *ProcessManager* component provides exactly those information needed for process enactment. This includes, for example, schemes of active process templates and running process instances as well as current instance states. In particular, *ProcessManager* restores instance-specific schemes for those process instances that were subject of previous ad-hoc changes. As opposed to *ProcessRepository*, *ProcessManager* has no knowledge about the evolution of process or activity templates; i.e., it is not aware of different template versions and their relations. This minimalism allows for efficient process enactment. As we discuss in Section 4, *ProcessManager* also deals with the migration of running process instances to a new process template version. It then has to interact with the *ProcessRepository* in order to retrieve the information required in this context; i.e., the schemes of the old and the updated process template as well as their difference.

***DataManager*:** For each process instance the *DataManager* maintains all process (relevant) data created during process enactment; i.e., all data elements and their values written by certain activities and read by other ones. Since process relevant data can become quite extensive and must be accessible by external components as well, they are not maintained within the *ProcessManager*, but through a separate component. The *DataManager* keeps all versions of a data element and creates a log entry each time the data element is accessed (in cooperation with the *LogManager*). Finally, the *DataManager* allows for implementing access functions for user-defined data types.

OrgModelManager and ***ResourceManager*** To define potential actors for a particular activity, it can be associated with an actor assignment. Such assignment refers to organizational entities (e.g., organizational units, project teams, roles, actors) or organizational relations (e.g., "is-manager-of") as captured in an organizational model. The *OrgModelManager* maintains this organizational model. It further accepts an actor assignment as input and delivers all actors qualifying for the respective expression as result. Besides actors, additional resources (e.g. rooms, machines and software licenses) can be maintained using the *ResourceManager* component.

3.2.3 Execution Layer

This layer comprises functional components of the ADEPT2 architecture which allow for the correct enactment and adaptation of process instances and related activities.

ChangeOperations: This component comprises high-level change operations that can be applied to processes in different context (e.g., to add or delete activities). First, change operations are required when creating new process templates or when adapting existing ones. In the latter case respective schema changes can be propagated to already running process instances as well; i.e., we (logically) apply the operations at instance level. The same applies with respect to ad-hoc instance changes. Note that in all these cases same or similar change operations are needed. Our basic design principles (cf. Section 3.1) therefore suggest to implement these change operations in a separate component to avoid code redundancies and to improve code maintainability. Each change operation realizes certain process graph transformations and is based on well-defined pre-/post-conditions in order to guarantee soundness of a process schema after its change. Note that conditions are varying depending on whether the change is applied at type or instance level.

ExecutionManager: This component coordinates the execution of process instances in an efficient and correct way; e.g., it evaluates predicates on instance data to choose between alternative branches or to loop back during runtime. As a prerequisite *ExecutionManager* needs information about the current schema as well as the state of respective instances. This information is provided by *ProcessManager*, i.e., a lower-layer component. For *ExecutionManager* it remains transparent whether a process instance is still running on its original schema or on a modified schema (as result of applied ad-hoc changes). When an activity is started the *ExecutionManager* provides the invoked application component with needed input data; when the activity completes, in turn, the *ExecutionManager* takes over its output data and forwards it to the *DataManager*.

RuntimeEnvironment: This component provides the container for executing arbitrary applications. It retrieves the input data of the respective application from the DataManager and prepares it for application invocation; i.e., the invoked application component does not need any knowledge about the specific process context in which it is executed. After completing an application execution successfully, in turn, the container receives the application output data and forwards it to the *DataManager*. Besides this, the *RuntimeEnvironment* allows to start application components and to control their execution (e.g., to abort or suspend component execution). Finally, the *RuntimeEnvironment* informs the ExecutionManager when the execution of an application fails.

3.2.4 User Interaction Layer

This layer comprises those components of the ADEPT2 architecture with which the different user groups interact. According to our basic philosophy all functions provided by these interactive components are made available via *application programming interfaces* (APIs) as well. This allows users to replace standard tools (e.g. for editing process templates or for managing user worklists) by own tool implementations.

ControlCenter: The ADEPT2 *ControlCenter* provides advanced buildtime and runtime components for user interactions. This includes the *ProcessEditor*, the *OrgModelEditor*, *Test Clients*, and the *Runtime Monitor*. The *ProcessEditor*, for example, constitutes the major component for modeling process templates and for guaranteeing model correctness (see Section 5). The *TestClient*, in turn, is a fully-fledged test environment for process execution. Unlike commonly known simulation tools, it runs on a lightweight instance of the process management system itself. As such, various execution modes between pure simulation and production mode are possible.

WorklistManager: This component manages worklists. When an activity becomes activated *WorklistManager* dissolves the corresponding actor assignment (in cooperation with *OrgModelManager*) and updates the respective worklists. The *WorklistManager* also considers deputy arrangements and allows to delegate work items to other users (even if the respective activity has been already started). Finally, escalation will be provided if a selected work item is not processed within a specified duration of time.

In summary, all described ADEPT2 system components are loosely coupled enabling the easy exchange of component implementations. Furthermore, basic infrastructure services

like storage management or the techniques used for inter-component communication can be easily exchanged. Additional plug-in interfaces are provided which allow for the extension of the core architecture, the data model and the user interface.

4 Architectural Support for Dynamic Process Changes in ADEPT2

So far we have introduced the ADEPT2 conceptual framework for dynamic process changes and we have sketched the different layers of the ADEPT2 system architecture. In this section we give insights into the realization of the ADEPT2 change framework within this architecture. Taking *process schema evolution* as example, we show in which way the different architectural components contribute to realize this feature and how they interact with each other to do this in a proper and efficient way.

4.1 A 2-phase procedure for realizing process schema evolution

When considering the ADEPT2 system architecture from Fig. 2 the general procedure for performing a *process schema evolution* comprises two phases (note that this procedure is simplified and does not consider interactions with lower-level services):

Phase I: Preparation Phase

1. Load an existing process template into the *ProcessEditor* and adapt its schema `S` using the change operations provided by *ChangeOperations.* Exactly the same set of change operations can be applied as when creating new process templates.
2. Record the modified process template (i.e., its target schema `S'`), together with the applied changes (i.e., the difference between `S'` and `S`), in the *ProcessRepository*.

Phase II: Schema Evolution Phase

3. Suspend (i.e. freeze) all process instances which are running on original process schema `S` and which shall be migrated to target schema `S'` (if possible).
4. Load target schema `S'` into *ProcessManager*. New instances are created based on `S'`.
5. Select original schema `S` and target schema `S'` in the *ProcessRepository* and transmit information about the schema difference `Delta` to the *ProcessManager.*
6. Based on `Delta`, for each frozen instance *ProcessManager* checks whether it is compliant with target schema `S'`. For this purpose *ProcessManager* considers the current instance state as well as instance-specific deviations from original schema `S`. The latter is required to detect conflicts between ad-hoc instance changes and the schema changes as captured by `Delta`.
7. *ProcessManager* migrates all compliant instances to target schema `S'`. Among other things this is accompanied by state adaptations of the instances to be migrated.
8. Where appropriate, adapted instances whose deviations conflict with process schema changes are adapted manually. This can be done using the components *ProcessEditor* and *ChangeOperations*. Again the migration is performed by *ProcessManager*.

This procedure already demonstrates that multiple system components are needed to enable process schema evolution in conjunction with other process support features.

4.2 How do architectural components of ADEPT2 support process changes?

For selected components of the ADEPT2 architecture we exemplarily show how they contribute to process flexibility in terms of schema evolution and ad-hoc changes. We revisit the described design principles and discuss their benefits in the given context.

***LogManager*:** Ad-hoc changes of single process instances as well as template changes have to be logged. The interfaces provided by the *LogManager* are generic; i.e., both kinds of changes can be logged with this component. Thus, LogManager can be reused in different context, which improves maintainability of the ADEPT2 architecture.

***ProcessRepository*:** If process schema evolution and instance migrations are supported we will have to maintain information about the different schema versions and their differences. This task is accomplished by the *ProcessRepository.*

***ProcessManager*:** This component is fundamental for the support of ad-hoc changes as well as of process schema evolution. It is therefore discussed in more detail. First, *ProcessManager* maintains the control data needed for proper and efficient execution of unchanged as well as changed process instances. Second, in the context of schema evolution, this component migrates compliant process instances to the new schema.

One major challenge is to efficiently represent template and instance objects within *ProcessManager*. Unchanged instances, for example, should be represented in a non-redundant way. The *ProcessManager* keeps one instance object for each of these unchanged instances, which captures instance-specific data (i.e., instance states) and refers to the original template schema (denoted as *template object* in the following). As example, consider instances I_1, I_3, I_4, and I_6 as depicted in Fig. 3.

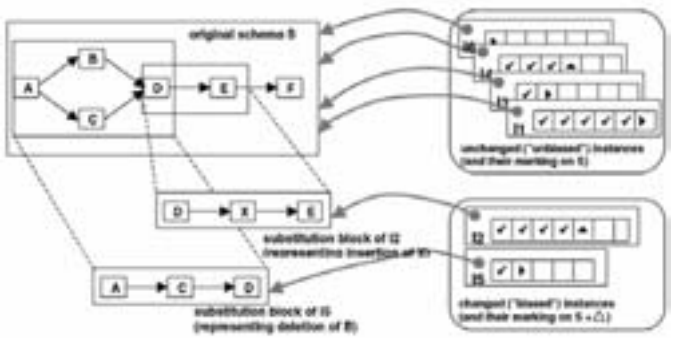

Figure 3: Managing Template and Instance Objects in the ProcessManager (Logical View)

For representing process instances with ad-hoc changes a more sophisticated approach is needed. In ADEPT2 we have developed the *delta layer concept* [Rind06, RJR07] for this

purpose. It allows to efficiently represent the difference between template and instance objects. Simply speaking, the delta layer is represented by an object with same interfaces as the process template object and therefore the same methods can be applied. However, a delta layer object does not reflect the whole process schema, but only those parts which have been adapted due to instance-specific changes. As examples consider instances I_2 and I_5 as shown in Fig. 3. Together with the template object the delta layer object allows to restore the instance-specific process schema. The instance objects which belong to changed process instances do no longer reference the associated template object but the delta layer object. The delta layer object itself references the original template object and therefore keeps the link between instance object and original template [Rind06].

The delta layer concept is also useful in the context of process schema evolution. In particular, it allows to quickly check whether instance-specific adaptations and template changes are conflicting with each other. Since *ProcessManager* supports ad-hoc changes anyway, schema evolution does not cause additional efforts when realizing this component. Note that we have decided to manage the different template versions and their *deltas* through a separate component (i.e., *ProcessRepository*). This historical information is only needed in the context of process schema evolution and should therefore not affect normal process enactment. (Here we assume that template changes constitute "exceptional cases" in comparison to normal process enactment.)

DataManager: To support instance-specific changes the *DataManager* must be able to dynamically add or delete process data elements. In this context, ADEPT2 deletes data elements and their values only logically in order to ensure traceability in all cases. Regarding schema evolution no additional functionality is required.

OrgModelManager: The support of template as well as instance changes imposes security issues as the process management system becomes more vulnerable to misuse. Therefore, the application of respective changes must be restricted to authorized users. We have developed an access control framework for (dynamic) process changes [WRW05] which can be based on the information managed by the *OrgModelManager* (see Section 3); i.e., similar to actor assignments specified in the context of process activities, we can define access control constraints for process changes (see [WRW05] for details). However, this requires no extensions of the *OrgModelManager* component. Currently, we are working on an advanced framework in order to enable evolving organizational models and the controlled adaptation of related actor assignments [RiRe07,RiRe08]. This new feature, in turn, will require extensions of *OrgModelManager;* e.g., the ability to maintain different versions of an organizational model or to adapt actor assignments on-the-fly when the underlying organizational model is changed.

ChangeOperations: As aforementioned this component allows to use the same change operations for modeling and adapting process templates as well as for defining instance-specific ad-hoc changes. As not all change operations might be needed in a given context, the set of accessible operations can be restricted. Furthermore, this component allows to add new change operations through well-defined interfaces. Finally, respective extensions do not influence the implementation of any other ADEPT2 component. This fundamental property is achieved by internally transforming high-level change operations into a basic set of stable change primitives (e.g., to add or delete nodes).

When modeling process templates structural schema changes are enabled by *ChangeOperations*. Regarding instance-specific changes, in addition, state adaptations become possible. Finally, process schema evolution requires the comparison of instance-specific changes with respective template changes conducted at the process type level. In our experience the complexity of these comparisons can be significantly reduced when using the delta layer concept as described before.

Schema evolution and instance-specific ad-hoc changes can be based on similar mechanisms. While for instance-specific adaptations the change operations and the respective state adaptations are applied in sequence, for schema evolution the structural changes and the subsequent state adaptations are applied all at once. In case of unchanged instances this only requires the re-linking of the instance objects to the new template object.

ExecutionManager: This component partially locks execution of running process instances when applying dynamic changes to them. After such change Execution-Manager re-evaluates the execution state of the modified instances in order to correctly proceed in the flow of control.

WorklistManager: *ExecutionManager* notifies *WorklistManager* when new activities become activated or running activities are completed. *WorklistManager* then updates user worklists accordingly; i.e., it adds new work items to worklists when enabling activities and removes items from them when completing activities. Basically, same functions can be used to adapt worklists when applying ad-hoc changes (e.g., for activating newly inserted activities) or when migrating process instances to an updated template.

RuntimeEnvironment: The *RuntimeEnvironment* only deals with the execution of single activities and related application components respectively. Therefore no specific functions with respect to schema evolution or instance-specific changes are needed.

ProcessEditor: To define template as well as instance-specific changes the *ProcessEditor* can be used (see Fig. 4). Among other features this component triggers the logging of the applied process changes. As aforementioned all functionality provided by *ProcessEditor* is made available via programming interfaces (APIs) as well.

Monitor: When changing an instance, which is currently displayed by the ADEPT2 monitor, the respective visualization is adapted automatically.

4.3 Proof-of-concept prototype

Except for the *ResourceManager* component, which will be added at a later stage of the project, we have implemented all components of the described architecture (cf. Fig. 2). In particular, our proof-of-concept prototype allows to demonstrate major flexibility concepts and their interplay with other process support functions. While core features of the components from the basic service layer and the execution layer have been implemented for most parts, we have not yet fully realized the intended tool components from the user interaction layer. For example, in the current release of the ADEPT2 system, ad-hoc changes of single process instances have to be accomplished using the ADEPT2 process editor. While the user interface provided by this editor is sufficient for

expert users, we will have to replace it if ad-hoc changes are to be defined by end users. Therefore, more sophisticated clients are under implementation. The same applies to clients enabling user interactions that become necessary to deal with non-decidable cases in the context of instance migrations. So far, ADEPT2 is able to automatically migrate all process instances that are compliant with the new schema version according to some correctness notion (see [RRD04b] for details). Furthermore, we have implemented a web client version of the WorklistManager. Regarding the low-level services layer, we have realized all basic functionality as needed by upper-layer components. However, there are still a lot of things to be done, e.g. concerning communication between system components and persistency management [Göse07]. Fig. 4 shows a screen of the ADEPT2 process editor, which constitutes the main system component for modeling and adapting process templates. This editor allows to quickly compose new process templates out of pre-defined activity templates, to guarantee schema correctness by construction and on-the-fly checks, and to integrate application components (e.g., web services) in a plug-and-play like fashion.

Another user component is the ADEPT2 Test Client. It provides a fully-fledged test environment for process execution and change. Unlike common test tools, this client runs on a light-weight variant of the ADEPT2 process management system. As such, various execution modes between pure simulation to production mode become possible.

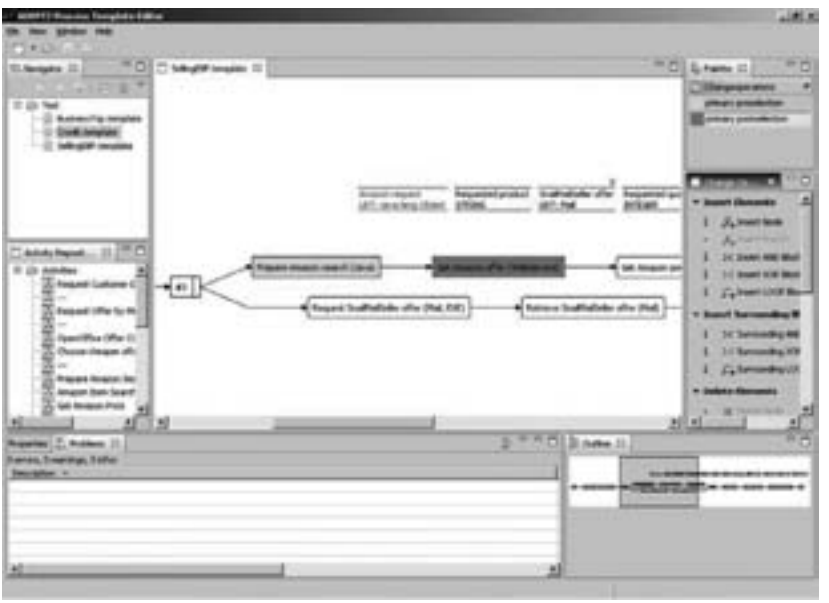

Figure 4: Screenshot of ADEPT2 Process Editor

Due to lack of space we cannot give a more detailed description of the provided functionality. We refer to [Göse07,ReDa98, RRD04b,WRR07] for a detailed overview of the concepts on which the implemented change features are grounded.

4.4 Summary

We have discussed how schema evolution and instance-specific changes have been considered in the ADEPT2 architecture. On the one hand we have shown that this architecture is able to cope with the different kinds of (dynamic) process changes. On the other hand, the given illustrations make clear that realization of schema evolution and ad-hoc changes within one system is far from being trivial. A proper system architecture with clear separation of concerns constitutes one necessary prerequisite in this context. Another one is a solid conceptual framework. When designing the ADEPT2 proof-of-concept prototype we have considered both perspectives. In future work we will evaluate the implemented concepts based on a series of lab experiments.

5 Related Work

Off-the-shelf process management systems like Staffware, FLOWer and IBM Process Server do not adequately support dynamic process changes or offer very restricted change features only [WRR07,WRR08]. Several vendors promise flexible process support, but are unable to cope with fundamental needs related to process change (e.g., correctness). Most systems completely lack support for ad-hoc changes or for migrating process instances to a changed process schema. Thus, application developers are forced to "enrich" applications with respective process support functions to cope with these limitations. This aggravates PAIS development and PAIS maintenance significantly.

The need for flexible and easily adaptable PAIS has been recognized and several competing paradigms for addressing process changes and process flexibility have been developed. Examples include adaptive processes [MGR04,MSK07,ReDa98,RRD04a+b, Wesk00], case handling [AWG05,MWR08], data-driven processes [MRH08], declarative workflows [Pesi07,SSO05], process refactoring [WeRe08] and late modeling [Adam06]. However, there is still a lack of implementations of respective technologies offering sufficient support to be applied for experimental use. Furthermore, only little work has been done with respect to the architectural design of respective systems considering requirements like extensibility, scalability, flexibility and maintainability.

Like ADEPT2, CAKE2 [MSK07] and WASA2 [Wesk00] allow for structural runtime adaptations at the process instance level. Both approaches only support change primitives (i.e., adding / removing nodes and edges respectively), while ADEPT2 provides support for a wide range of high-level change operations [WRR07]. ADEPT2 is the only system which provides common support for both process schema evolution and ad-hoc changes [WRR07,RRD04a]. Worklets [Adam06] allow for the late binding of sub-processes following a rule-based approach. Except for the dynamic replacement of activities no support for ad-hoc changes is provided. Similar considerations can be made for the case handling tool FLOWer [AWG05.MWR08], which allows to delete activities, but does not support other kinds of ad-hoc changes. Neither Worklets nor FLOWer have considered issues related to process schema evolution. Finally, among all these approaches ADEPT2 scores best in respect to high-level change operations [WRR07,WRR08].

6 Summary

The ADEPT2 technology meets major requirements claimed for next generation process management technology. It provides advanced functionality to support process composition by plug & play of arbitrary application components, it enables ad-hoc flexibility for process instances without losing control, and it supports process schema evolution in a controlled and efficient manner. As opposed to other approaches all these aspects work in interplay as well. For example, it is possible to propagate process schema changes to individually modified process instances or to dynamically compose processes out of existing application components. All in all such a complex system requires an adequate conceptual framework and a proper system architecture. ADEPT2 is one of the very few systems which has tried to consider both conceptual and architectural considerations in the design of a next generation process management system.

References

[Adam06] Adams, M.; ter Hofstede, A.H.M.; Edmond, D.; van der Aalst,W.: A service-oriented implementation of dynamic flexibility in workflows. Proc. Coopis'06 (2006)

[AWG05] van der Aalst, W.; Weske, M.; Grünbauer, D.: Case handling: a new paradigm for business process support, Data and Knowledge Engineering. 53 (2) (2005) 129-162.

[BDR03] Bauer, T.; Reichert, M.; Dadam, P.: Intra-subnet load balancing in distributed workflow management systems. Int'l Journal of Cooperative Information Systems , 12(3):205-323, 2003

[DRK00] Dadam, P.; Reichert, M.; Kuhn, K.: Clinical workflows – the killer application for process-oriented information systems? Proc. 4th Int'l Conf. on Business Information Systems (BIS'2000), Poznan, Poland, April 2000, pp. 36-59.

[Göse07] Göser, K. et al.: Next-generation process management with ADEPT2. Proc. of the BPM'07 Demonstration Program, Brisbane, Australia, September 2007, pp. 3-6.

[Guen06] Guenther, C.W.; Rinderle, S.; Reichert, M.; van der Aalst, W.M.P. (2006) Change mining in adaptive process management systems. In: Proc. 14th Int'l Conf. on Coop. Information Systems (Coopls'06), 2006, Montpellier. LNCS 4275, pp. 309-326

[LeRe07] Lenz, R.; Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives, Data and Knowledge Engineering, 61(1):39-58, 2007.

[LRW08] Li, C.; Reichert, M.; Wombacher, A.: Discovering reference process models by mining process variants. In: Proc. 6th Int'l Conf. on Web Services (ICWS'08), September 2008, Beijing, China. IEEE Computer Society Press

[MGR04] Müller, R.; Greiner, U.; Rahm, E.: AgentWork: A workflow system supporting rule-based workflow adaptation, Data and Knowledge Engineering 51 (2) (2004) 223-256.

[MSK07] Minor, M.; Schmalen, D.; Koldeho, A. workflow supported by a suspension. Proc. WETICE'07, 2007.

[MRB08] Mutschler, B.; Reichert, M.; Bumiller, J.: Unleashing the effectiveness of process-oriented information systems: problem analysis, critical success factors and implications. IEEE Transactions on Systems, Man, and Cybernetics, 38 (3):280-291, 2008.

[MRH08] Müller, D.; Reichert, M.; Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. Proc. 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), Montpellier, LNCS 5074, pp. 48-63

[Müll06] Müller, D.; Herbst, J.; Hammori, M.; Reichert, M.: IT Support for release management processes in the automotive industry. In: Proc. 4th Int'l Conf. on Business Process Management (BPM'06), Vienna, Austria. LNCS 4102, pp. 368-377

[MWR08] Mutschler, B.; Weber, B.; Reichert, M..: Workflow management versus case handling: results from a controlled software experiment. Proc. 23rd Annual ACM Symposium on Applied Computing (SAC'08), Fortaleza, Ceará, Brazil, March 2008, pp. 82-89

[Pesi07] Pesic, M.; Schonenberg, M.; Sidorova, N.; van der Aalst, W.M.P.: Constraint-based workflow models: change made easy, Proc. CoopIS'07 Conf., 2007.

[RDB03] Reichert, M.; Dadam, P.; Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Int'l Journal Software and Systems Modeling (SOSYM), 2(1):37-58, 2003

[ReDa98] Reichert, M.; Dadam, P.: ADEPTflex – Supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems, 10(2):93-129, 1998

[Rind06] Rinderle, S.; Reichert, M; Jurisch, M.; Kreher, U.: On representing, purging and utilizing change logs in process management systems. Proc. 4th Int'l Conf. Business Process Management (BPM'06), Vienna, LNCS 4102, September 2006, pp. 241-256

[RiRe07] Rinderle, S.; Reichert, M.: A formal framework for adaptive access control models. Journal on Data Semantics IX, LNCS 4601, Springer 2007, pp. 82-112.

[RiRe08] Rinderle-Ma, S.; Reichert, M.: Managing the Life Cycle of Access Rules in CEOSIS. Proc. 12th IEEE Int. Enterprise Computing Conference (EDOC'08), September 2008, Munich, Germany.

[RJR07] Rinderle, S.; Jurisch, M.; Reichert, M.: On Deriving Net Change Information From Change Logs – The DELTALAYER-Algorithm. Proc. BTW'07, 2007, pp. 364-381

[RRD03a] Reichert, M.; Rinderle, S.; Dadam, P.: On the common support of workflow type and instance changes under correctness constraints. In: Proc. 11th Int'l Conf. Cooperative Information Systems (CooplS '03), Catania, Italy. LNCS 2888, pp. 407-425

[RRD03b] Reichert, M.; Rinderle, S.; Dadam, P.: ADEPT Workflow Management System Flexible Support for Enterprise-Wide Business Processes. Proc. 1st Int'l Conf. on Business Process Management (BPM '03), Eindhoven, LNCS 2678, pp. 371-379

[RRD04a] Rinderle, S.; Reichert, M.; Dadam, P.: Correctness criteria for dynamic changes in workflow systems - a survey. Data and Knowledge Engineering, 50(1):9-34 (2004)

[RRD04b] Rinderle, S.; Reichert, M.; Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases, 16(1):91-116, 2004

[RRD04c] Rinderle, S.; Reichert, M.; Dadam, P.: On Dealing with structural conflicts between process type and instance changes. In: Proc. 2nd Int'l Conf. Business Process Management (BPM'04), Potsdam, Germany. LNCS 3080, 2004, pp. 274-289

[RRD04d] Rinderle, S., Reichert, M.; Dadam, P.: Disjoint and overlapping process changes: challenges, solutions, applications. In: Proc. 11th Int'l Conf. on Cooperative Information Systems (CooplS'04), Agia Napa, Cyprus. LNCS 3290, 2004, pp. 101-121

[SSO05] Sadiq, S.; Sadiq, W.; Orlowska, M.: A framework for constraint specification and validation in flexible workflows. Information Systems 30, 349-378, 2005

[Wesk07] Weske, M.: Business Process Management, Springer, 2007.

[Wesk00] Weske, M.: Workflow Management Systems: Formal foundation, conceptual design, implementation aspects. Habilitationsschrift, University of Münster, 2000

[WRW05] Weber, B. ; Reichert, M. ; Wild, W. ; Rinderle, S.: Balancing flexibility and security in adaptive process management systems. Proc. CoopIS'05, LNCS 3760, pp. 59-76

[WRR07] Weber, B.; Rinderle, S.; Reichert, M.: Change patterns and change support features in process-aware information systems. Proc. 19th Int'l Conf. on Advanced Inform. Sys. Engineering (CAiSE'07), LNCS 4495, Trondheim, June 2007, pp. 574-588

[WRR08] Weber, B.; Reichert, M.; Rinderle-Ma, S. (2008) Change patterns and change support features - enhancing flexibility in pocess-aware information systems. Data and Knowledge Engineering (accepted for publication)

[WeRe08] Weber, B.; Reichert, M.: Refactoring process models in large process repositories. Proc. 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), June 2008, Montpellier, France, LNCS 5074, pp. 124-139.

An Environment for Modeling Workflow Components

Colin Atkinson and Dietmar Stoll

Lehrstuhl für Softwaretechnik
University of Mannheim
68131 Mannheim
{atkinson, stoll}@informatik.uni-mannheim.de

Abstract: An important goal of workflow engines is to simplify the way in which the interaction of workflows and software components (or services) is described and implemented. The vision of the AristaFlow project is to support a "plug and play" approach in which workflow designers can describe interactions with components simply by "dragging" them from a repository and "dropping" them into appropriate points of a new workflow. However, to support such an approach in a practical and dependable way it is necessary to have semantically rich descriptions of components (or services) which can be used to perform automated compatibility checks and can be easily understood by human workflow designers. This, in turn, requires a modeling environment which supports multiple views on components and allows these to be easily generated and navigated around. In this paper we describe the Integrated Development Environment (IDE) developed in the AristaFlow project to support these requirements. After outlining the characteristics of the "plug and play" workflow development model, the paper describes one of the main innovations within the IDE –the multi-dimensional navigation over views.

1 Introduction

An important goal of workflow engines is to simplify the way in which the interaction of processes and software components (or services) is described and implemented [DR04, Ac04]. The AristaFlow project's vision of how to achieve this is based on the "plug and play" notion popularized on the desktop, in which workflow designers can describe interactions with components simply by "dragging" them from a repository and "dropping" them into the desired points of a new workflow [Da05]. However, the ability to define new workflows in such a simple and straightforward way is only advantageous if there is a high likelihood that the resulting processes are well-formed, correct and reliable. In other words, to make the "plug and play" metaphor work in practical workflow scenarios it is essential that components are used in the "correct way", and the possibility for run-time errors is significantly reduced at design time. In short, there should be few if any "surprises" at run-time. If workflows defined by the "plug and play" metaphor are highly unreliable or unpredictable this approach will not be used in practice.

In order to support this goal, components must be described in a way that –

1. has well defined semantics so that their properties are machine-readable and can be used to automatically check workflow-component compatibility, correctness and reliability.

2. is easy for humans to understand, so that workflow designers can easily comprehend components' properties and decide which components to use where and in what way.

Only component description approaches that fulfill both of these requirements provide the required foundation for the "plug and play" development and adaptation of workflows. In addition, of course, workflow components must be developed using the best available practices and subject to rigorous validation and quality assurance activities (e.g. inspection and testing). So called "Semantic" approaches for describing components/services, such as OWL-S [Ow04] or WSMO [Ws05] score highly on the first requirement since they utilize a description logic based language such as OWL to describe component semantics in a rigorous and machine-accessible way. However, since they are optimized for reasoning efficiency rather than human readability they are difficult to use.

Model-based representations of components based on languages such as the UML score much more highly on the second requirement, but score less well on the first requirement. This is because the semantics of some of the UML diagrams is somewhat vague, and it is unclear what combination of diagrams should be used to fully document a component and what information each diagram should contain. Indeed, the views supported by the current set of UML diagrams do not allow all the necessary information to be described and/or do not present it in an appropriate way. Moreover, there are no predefined relationships between the UML diagram types, so there is no built-in way of determining whether different views of a component are consistent with one another.

Nevertheless, modeling languages such as the UML provide a much more suitable foundation for describing workflow components in a way that supports the plug and play paradigm than OWL based approaches. By using OCL to tighten the semantics of models and adding additional view types optimized for workflows it is possible to overcome these problems and attain a component/service representation approach which fulfills both criteria outline above. However, to make this viable in practice it is necessary to define suitable consistency rules between views and provide a pragmatic metaphor for creating and navigating around them. In addition, the approach must be integrated within a practical software engineering environment that allows components to be designed, implemented and tested using traditional development techniques.

This paper describes the approach to component modeling and development within the AristaFlow project and the integrated development environment (IDE) created to support it. Although these are optimized for the description of workflow components (e.g. by views and editors especially tailored to the requirements of workflow developers and administrators), they are useful for general component modeling as well. In the next chapter we describe the overall life-cycle of components, and describe how they fit into

the workflow definition and execution process. The following two sections then describe the main innovations in the AristaFlow approach. Chapter 3 describes the AristaFlow IDE's strategy for integrating the various kinds of diagrams types and view types needed to fully describe workflow components and for ensuring that they stay consistent. Chapter 4 describes the IDE's innovative strategy for organizing the different views and supporting navigation around them. Chapter 5 then presents some implementation details of the IDE, and chapter 6 concludes with some final remarks.

2 Component-Oriented Development of Workflows

The AristaFlow project aims to cover the whole lifecycle of components from their initial development to their use in workflow management systems (Figure 1). In this lifecycle there are three main human roles: the component developer, the workflow administrator and the workflow developer.

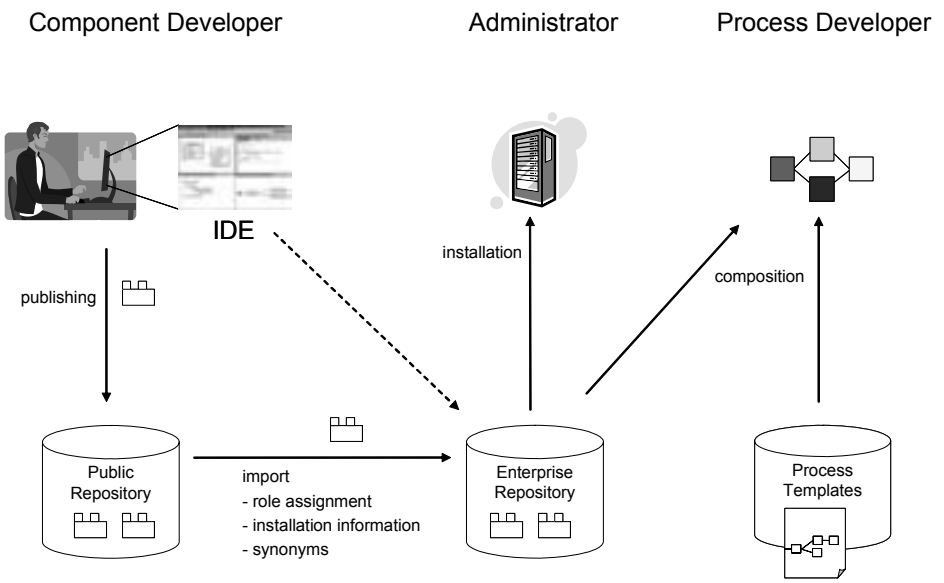

Figure 1: Workflow Component Lifecycle

The component developer models and implements components and then publishes them in a public repository. A workflow administrator of an enterprise then browses the repository or searches it by using various criteria as defined in the component description. Once they have been found, suitable components can be imported into a private enterprise repository – the so called deployment store – where they are installed and ready to be executed. The administrator can then add information like role assignments and deployment information. In addition, taxonomical information can be adjusted or added, e.g. synonyms and (enterprise wide) unique identifiers for parameters,

and these can later be automatically "wired" to data elements by the workflow management system.

A process developer then uses a process template editor to combine components and process templates into executable processes. Figure 2 shows a screenshot of the ADEPT2 editor developed in the AristaFlow project [Ar07]. In this example, an application function ("Amazon Item Search") has been chosen from the activity repository and dragged into the process graph. As the input parameter is not yet assigned to another node in the process graph, the problem window shows an error message. A process template can only be released to the ADEPT2 runtime system (for later instantiation) if it contains no errors. Similar checks take place at runtime, when ad-hoc changes or process schema evolutions are made [RD03, Ri04]. The system only allows changes if they do not lead to inconsistencies. An extensive description of the issues and requirements involved in modern workflow process modeling tools and workflow management systems such as ADEPT2 is given in [Da05].

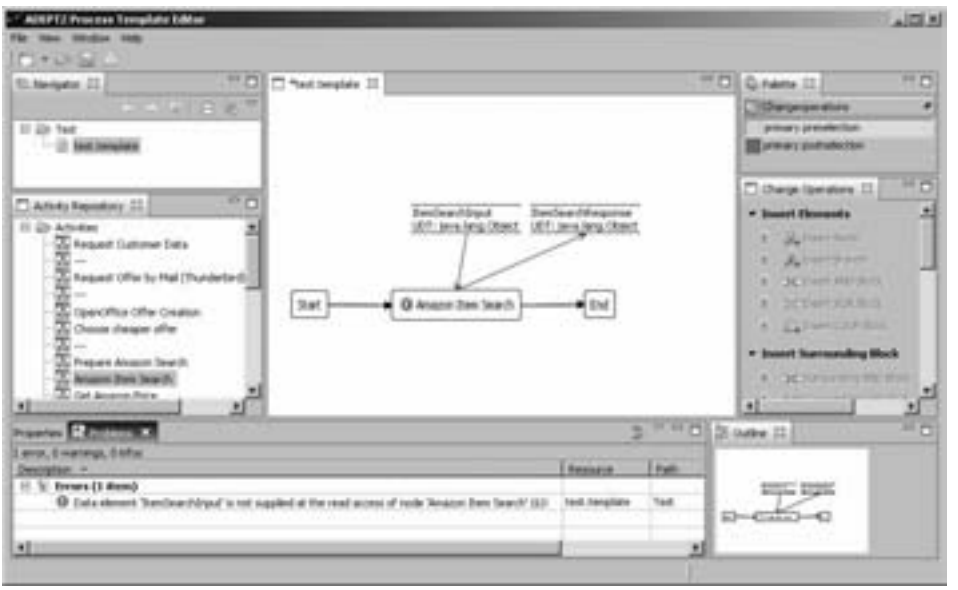

Figure 2: AristaFlow Workflow Editor

2.1 Integrated Development Environment

Since components/services are software applications in their own right, a component development IDE needs to support the full range of development activities including code development, testing and debugging. However, when building a component IDE it is clearly undesirable and impractical to redesign or redevelop the rich range of capabilities that modern IDEs provide. The AristaFlow IDE is therefore built upon an existing, well known and extensible development environment - namely the Eclipse environment.

Taking this goal into account, the component modeling approach and IDE were developed to fulfill the following main requirements –

1. to fit seamlessly and with minimum impact on top of the Eclipse environment, giving developers access to the full range of native Eclipse functionality and existing Eclipse plug-ins,

2. to provide a concise, well-defined and human friendly representation of components which workflow designers can easily understand and use to select and employ components in their workflows,

3. to provide suitable, machine readable descriptions of component properties which automated checkers (e.g. the workflow editor, the workflow execution engine) can use to verify the suitable use of components within workflows, and

4. to support the packaging (exporting) of components in a way that can easily be imported by the repository and the workflow tools.

Given requirements (1) and (3), it makes sense for the AristaFlow IDE to exploit as much of the information in regular software development artifacts as possible and translate it automatically into formats that can be understood by workflow management systems. For example, activity templates, which are required for process modeling, can to a large extent be automatically generated from source code. If the parameters of operations of different vendors are related (e.g. by two parameters that represent an account number), a mapping between parameters and unique identifiers is defined to enable the automatic association of data elements to parameters. It is also possible to associate a component or single operation with one or more taxonomies. The taxonomy editor of the IDE allows custom taxonomies to be imported or created.

When a component developer specifies the behaviour of a component, for example, as a state chart whose transitions correspond to operation calls or as regular expressions, protocol checkers can be employed. These make sure that a process modeler can plug operations into a workflow schema only in a way that obeys the constraints specified by the component modeler. Similarly, constraints (i.e. invariants and pre- and post conditions on operations) can be used for runtime checks, for example, by arranging for the workflow execution engine to check preconditions before an operation is actually invoked. The code for checking the preconditions can be generated by the IDE and delivered with the component. This makes it possible to catch violated preconditions that could lead to expensive and difficult-to-trace runtime errors and to generate a response understandable to a workflow administrator.

The more information that is available to the execution environment, the more checks can be made. Many of the checks are done at modeling time on the process schemata, ensuring incorrect schemata are not allowed to be executed. In cases where modeling time checks are impossible or too complex, checks at runtime are applied. These can prevent unwanted behaviour of components, for example, by checking whether the preconditions of an operation have been fulfilled by the component developer before it is called.

3 View-Based Component Modeling

Although the basic idea of capturing software artifacts from numerous inter-related viewpoints has been around for some time (e.g. [NKF03]), there are no widely used tools that provide clean and inherent support for this approach. With the success of the agile development movement most applications are still developed using source code as the single view of software objects, and although the use of multi-view notations such as the UML has grown in popularity, the selection of which views to use and how views should be related is invariably left to the user. In particular, there are no widely used component modeling tools that provide users with the flexibility to define new view types and generate/access new view instances on demand while at the same time systematically enforcing and checking consistency rules.

The problem is that during the development of a component, users need to generate and work on all kinds of views ranging from UML diagrams to code fragments, and as the number of views increases, navigation becomes more tedious and maintaining consistency between them becomes increasingly difficult. This is particular so when the relationships and consistency rules between different types of views are defined and checked on a pairwise basis as is usually the case today.

An ideal solution to this problem would be for every view of the IDE to be generated from, and to work on, a single underlying model and for changes made to individual views to be synchronized directly with this model [AS07]. In this way, the consistency between the editors and the model is automatically ensured, as long as each individual change to a view is checked for validity against this model. This "on demand" generation of views is schematically depicted in Figure 3. The single underlying model should be an instance of a metamodel which contains the minimum set of concepts necessary to store the required information.

However, building the whole IDE in this way, although possible in the long term, is incompatible with requirement (1) in the short term. Thus, in the AristaFlow project a hybrid solution was developed in which several underlying model formats coexist. In the long term these will be merged into one representation and all views will be generated (by model transformation) from this single representation.

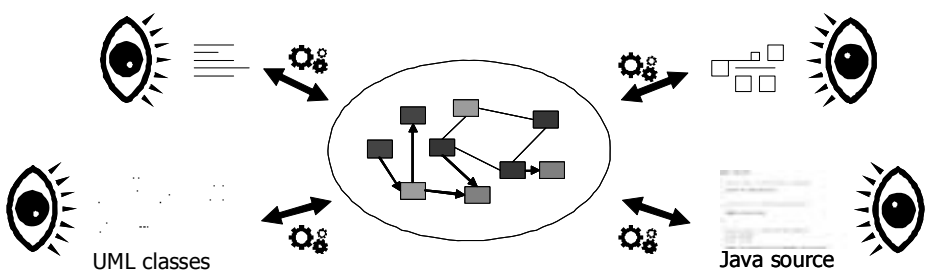

Figure 3: On demand View Generation

3.1 View Types

Although advanced users need to have the ability to define their own view types and to describe how they relate to the existing views, most users will be content with using the existing view types defined by the recommended modeling approach. A key question, therefore, is what set of basic views users should employ to describe components. AristaFlow's basic approach to component modeling is based on the KobrA approach [At02], which defines a systematic approach to the UML based representation of components. This is organized around the notion of **projections**, which define the kind of information conveyed in a view, and **abstraction levels**, which define the level of platform independence represented by a view and whether the information conveyed is black box or white box.

There are currently three levels of abstraction supported in the IDE: specification, realization and implementation. The most abstract level is the specification which provides a black box view of the component. It describes all externally visible properties of a component and thus serves as its requirements specification. The realization of a component describes the design of the component and provides a white box view of its internal algorithms and subcomponents. Source code and test cases are the most platform specific representation, and capture the implementation of the component using the chosen programming language.

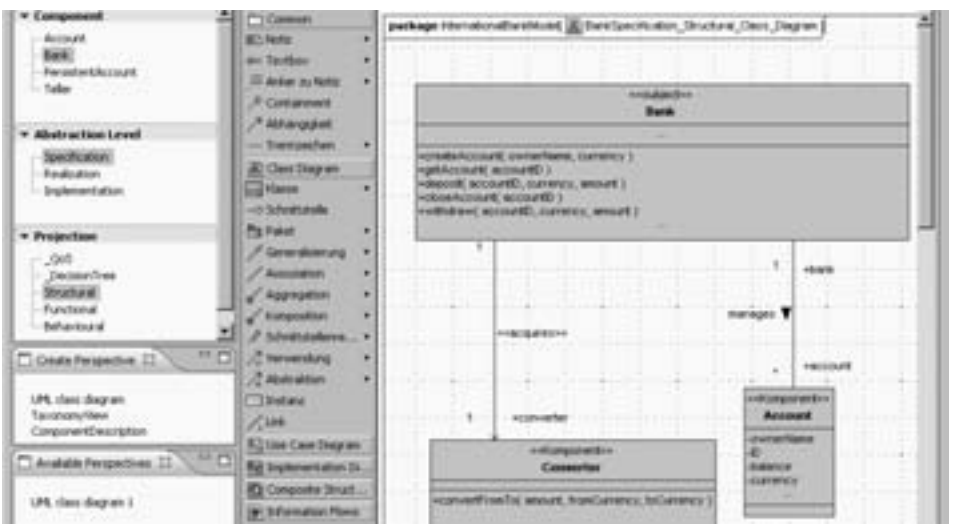

Figure 4: Structural view of a Component Specification

KobrA also defines three fundamental projections: the structural, functional and behavioural projection [At02]. The structural projection includes classes and associations manipulated by the component as well as other structural information like taxonomical information and source code. Operations of a component and their interaction with other artifacts are modeled in the functional projection, e.g. by means of operation specifications and UML interaction diagrams. Finally, the behavioural projection focuses

on the behaviour of the component and its operations, as manifest by UML state charts and UML activity diagrams.

The basic principle behind KobrA is that a component should be viewable and describable at both the specification and realization levels of abstraction from all three "projection" perspectives. Thus, the specification of a component can be viewed from a structural, functional and behavioural viewpoint, and the realization of a component can be viewed from a structural, functional and behavioural viewpoint. Figure 4, for example, is a screenshot from the IDE which shows the structural view of the specification of a Bank component. This takes the form of a UML class diagram which shows only externally visible properties of the component and its environment.

Figure 5, on the other hand, is a screenshot of the IDE which shows an element of the functional view of the Bank specification. This is a so called "operation specification" for an operation of the Bank (the withdraw operation), and defines the effects of the operation in terms of OCL pre and post conditions. It is only one element of the

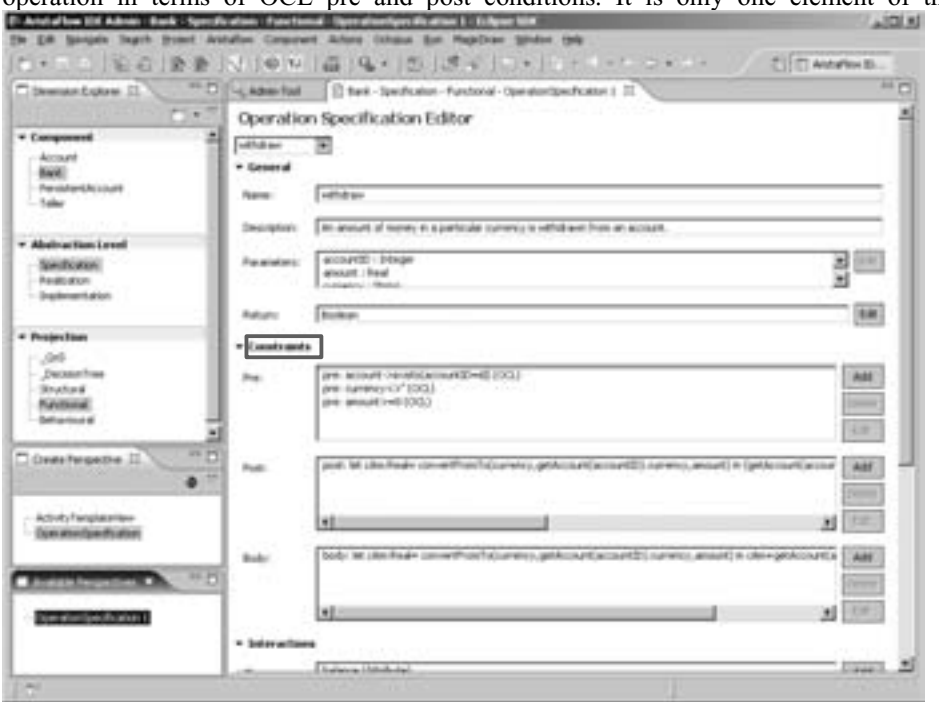

Figure 5: Functional View of a Component Specification

functional view of the Bank specification, because such a specification is needed for each operation. Similarly, the behavioural view of a component specification consists of a UML state diagram depicting the externally visible state and transitions of the component. However, this is not shown here for space reasons.

These views are general views on components defined in the KobrA method. With one exception they all employ a UML diagram. The exception is the operation specification which is a form based view that uses the OCL. However, to support requirement (3) above, additional views are supported in the IDE in order to provide information directly needed for workflow compatibility checking. For example, Figure 6 below shows a view which is used to define the classification of a component within a standardized business taxonomy (like UNSPSC [Un07] or NAICS [Na07]). This provides information which is directly used by the repository to support the cataloguing and organized browsing of components.

Figure 6: Component Classification View

Figure 7 shows an even more detailed view which allows users to add extra information needed explicitly by the ADEPT workflow editor and execution system.

Additional views can also obviously be added for different purposes. For example, it would be easily possible to add higher-level business oriented views such as those defined by [Tu02].

Of course, to support the integration of legacy components and not overwhelm developers with features they are not familiar with, a developer is not forced to use all the possibilities of the development environment. Only the use of a minimal set of artifacts for the component repository is mandatory, e.g. information about the executable operations of a component.

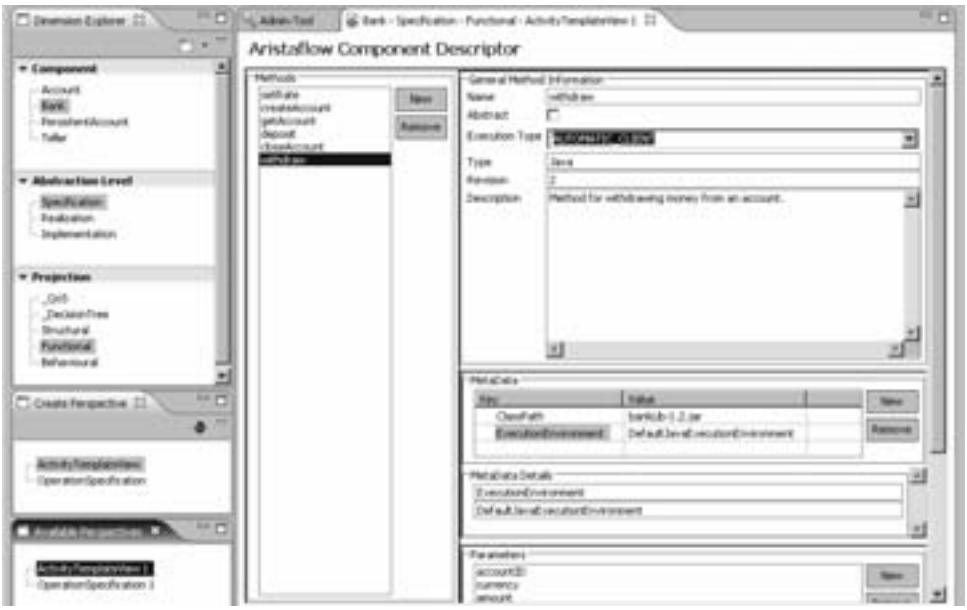

Figure 7: Component Descriptor

4 Dimension-based Navigation

Supporting a fundamentally view-based way of creating and manipulating components of the form described in the previous section can greatly simplify the task of developing components and assessing whether they are suitable for use in workflows. Different stakeholders can view a component using diagram types and notations which best meet their needs and expertise, and specialized views can be generated for specific purposes. However, the downside to a view-based approach is that the number of views can quickly explode. As a result, the benefit gained by the simplicity and clarity of individual views can be outweighed by the extra complexity and overhead involved in organizing and navigating around a large number of different views. This problem is particularly acute in environments which use different, third party editors to generate and manage views, since a user must then become acquainted with and navigate around different, heterogeneous artifact trees.

An effective view-based IDE should therefore provide a simple, integrated approach for managing and navigating around the various views supported by the system. To meet this need the AristaFlow IDE employs a new navigation metaphor based on the notion of independent, orthogonal development dimensions. This is motivated by the "orthographic projection" paradigm used in mechanical and physical engineering to create detailed drawings of physical objects, and exploits the fact that the different projections and abstract levels used in KobrA to define the different views are essentially

orthogonal and hence can be selected independently. This is no accident, since the KobrA method explicitly recognizes the existence of three fundamental and orthogonal development dimensions. However, the use of these as a navigation metaphor is original in the AristaFlow IDE. This is why we use the name "orthographic modeling" to characterize the representation approach supported by the IDE.

In principle, there is no limit to the number of dimensions that can be supported. However, in the current version of the IDE there are three dimensions: an abstraction level dimension which represents the abstraction level discussed in the previous section and has three distinct choices (specification, realization and implementation), a projection dimension which represents the projection type discussed in the previous section and also has three distinct choices (structural, functional and behavioural), and a component dimension, which represents the component which is being worked on or viewed. This has as many choices as there are components in the system – one for each component.

The organization of the views around the notion of three orthogonal dimensions can be visualized in terms of a cube, as illustrated in Figure 8. Each view corresponds to a cell in the cube, which represents a particular choice for each of the independent dimensions. Users are thus able select particular views by navigating around the cube and selecting specific cells corresponding to specific choices of component, abstraction level and projection.

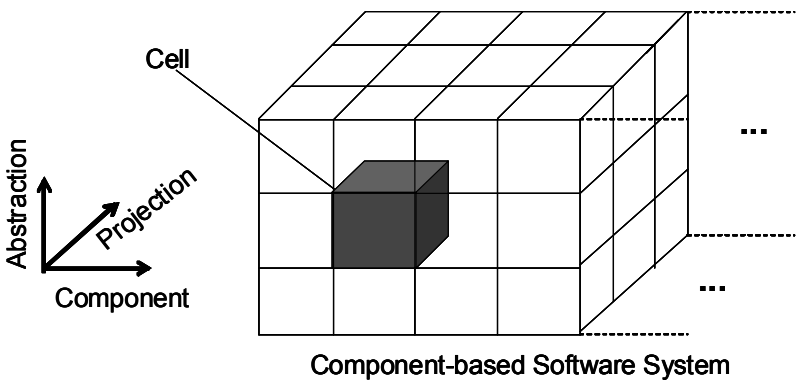

Figure 8: Cube metaphor

Figure 4 to 7 show how this dimension-based navigation is actually supported in the current IDE. The left hand side of each of these diagrams shows the navigation area which contains a selection panel for each of the three dimensions, each showing the currently selected option for each dimension. Thus, the navigation area on the left hand side of Figure 4 shows that the displayed UML diagram actually occupies the cell corresponding to the Bank option of the component dimension, the specification option of the abstraction-level dimension and the structural option of the projection dimension.

In other words, it shows a structural view of the specification of the Bank component. Similarly, the navigation area on the left hand side of Figure 5 shows that the displayed operation specification occupies the cell corresponding to the Bank option of the component dimension, the specification option of the abstraction level dimension and the functional option of the projection dimension. In other words, it shows a functional view of the specification of the Bank component. Obviously, by selecting different combinations of choices from each dimension, users can navigate to different views.

Usually, one cell is associated with exactly one editor, e.g. a UML class diagram is associated with the UML tool MagicDraw. However, if greater flexibility is desired, a cell can be mapped to multiple editors, for example, when there are alternative tools available for UML class diagrams. This is the role of the bottom selection panel. It identifies which specific representation or rendering of a view is desired.

5 Configuration of the IDE

As mentioned above, to create a practical prototype IDE within the original AristaFlow project a number of existing editors and tools were integrated under the view-based metaphor just described. In this section we briefly explain what tools were integrated and what role they play.

5.1 Editor overview

Projection / Abstraction	Structural	Functional	Behavioural
Specification	UML Class Diagram Taxonomy Component Descript.	Operation Specification Activity Template	UML State Chart Regular Expression
Realization	UML Class Diagram	UML Communication Diagram	UML Activity Diagram
Implementation	Source Code	-	-

Figure 9: Overview of editors for workflow component modeling

As an elegant and widely available UML diagramming tool, MagicDraw was chosen for the following Perspectives: Class Diagram, State Chart, Communication Diagram, and Activity Diagram. The Class Diagram editor is used for both the black box view (*Structural – Specification*) and the white box view (*Structural – Realization*) of a component. The behaviour of a component is modeled with a UML State Chart. In the Structural Realization, the UML class diagram from the Structural Specification is refined. The Functional Realization shows by the means of a UML communication

diagram with which components the function interacts. The Behavioural Realization focuses on the decomposition of functions by exposing the internal logic with a UML activity diagram.
The Operation Specification Editor focuses on pre- and post conditions of single operations and their syntactical correctness. Especially suited for the component repository are the Component Description Editor, the Taxonomy Editor, the Activity Template Editor and the Regular Expression Editor (whose description can be used for checks of allowable method call sequences at run-time).

After implementing a component, all artifacts can be packaged and saved in a single file in the Reusable Asset Specification (RAS) format. The RAS is an OMG standard specifying the structure, contents and description for reusable software components/assets [Ras05]. Thus, a RAS can be populated with all the artifacts generated during the component development process including models, requirement specifications and tests as well as the final source code. This allows the component to be imported into arbitrary development environments (i.e. any IDE that supports the RAS) for further development and maintenance when the need arises.

6 Conclusion

The goal of the AristaFlow project was to develop and prototypically implement a platform to support the whole lifecycle of flexible, process-aware information systems - from the modeling and implementation of suitable components through process composition in a plug and play like fashion up to flexible and adaptive process enactment. An important part of this goal was to minimize errors at runtime by using advanced component development and process composition methods. The IDE and component repository described in this paper were developed to support the component modeling element of this concept.

The developed IDE makes two major contributions to the state of the art – the first is a novel approach for the dynamic generation of views on demand, and the second is a novel approach for allowing users to organize and navigate around the different views. While the IDE was specifically developed for the AristaFlow project, its view, navigation and component representation concepts are useful for other software development approaches as well.

Acknowledgements**:** This work was largely performed as part of the AristaFlow project under the support of the State of Baden-Württemberg.

References

[Ac04] H. Acker, C. Atkinson, P. Dadam, S. Rinderle, M. Reichert: *Aspekte der komponentenorientierten Entwicklung adaptiver prozessorientierter Unternehmenssoftware*. In: K. Turowski (Hrsg.): Architekturen, Komponenten,

Anwendungen - Proc. 1. Verbundtagung AKA 2004, Augsburg, Dezember 2004. LNI P-57, 2004, S. 7-24

[At02] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, J. Zettel: Component-Based Product Line Engineering with UML. Addison-Wesley Publishing Company, 2002

[Ar07] The AristaFlow project, http://www.aristaflow.de, visited Nov 2007

[AS07] C. Atkinson, D. Stoll: *Orthographic Modelling Environment*, Fundamental Approaches to Software Engineering (FASE'08), Budapest (Hungary), 29 March - 6 April, 2008, submitted

[Da05] P. Dadam, M. Reichert, S. Rinderle, C. Atkinson: *Auf dem Weg zu prozessorientierten Informationssystemen der nächsten Generation - Herausforderungen und Lösungskonzepte*. In: D. Spath, K. Haasis, D. Klumpp (Hrsg.): Aktuelle Trends in der Softwareforschung - Tagungsband zum doIT Software-Forschungstag 2005, Karlsruhe, Juni 2005. Schriftenreihe zum doIT Software-Forschungstag, Band 3, MFG Stiftung, 2005, S. 47-67

[DR04] P. Dadam, M. Reichert: *ADEPT - Prozess-Management-Technologie der nächsten Generation*. In: D. Spath, K. Haasis (Eds.): Aktuelle Trends in der Softwareforschung - Tagungsband zum doIT Software-Forschungstag 2003, IRB Verlag Stuttgart 2004, S. 27-43

[Na07] *The North American Industry Classification System* (NAICS), http://www.census.gov/epcd/www/naics.html, visited May 2007

[NKF03] B. Nuseibeh, A. Finkelstein and J. Kramer, "ViewPoints: meaningful relationships are difficult," International Conference on Software Engineering (ICSE 2003), Portland, Oregon, 2003

[Ow04] The OWL Services Coalition, OWL-S 1.1 Release, http://www.daml.org/services/owl-s/1.1/, visited Nov 2007

[Ri05] S. Rinderle: Schema Evolution in Process Management Systems. Dissertation, Universität Ulm, Fakultät für Informatik, Dezember 2004

[RD03] S. Rinderle, P. Dadam: *Schemaevolution in Workflow-Management-Systemen* ("Aktuelles Schlagwort"). Informatik-Spektrum, Band 26, Heft 1, Februar 2003, S. 17-19

[Ras05] *Object Management Group, Reusable Asset Specification, version 2.2*. http://www.omg.org/technology/documents/formal/ras.htm, Nov 2005, visited May 2007

[Tu02] K. Turowski (Editor) et al., *Standardized Specification of Business Components*, Memorandum of the working group 5.10.3 Component Oriented Business Application System, February 2002, http://www.wi2.info/downl/gi-files/MEMO/Memorandum-english-final-included.pdf, visited Nov 2007

[Un07] *The United Nations Standard Products and Services Code (UNSPSC)*, http://www.unspsc.org/, visited May 2007

[Ws05] Web Service Modeling Ontology (WSMO), W3C Member Submission, http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/, visited Nov 2007

A Pragmatic Approach to Traceability in Model-Driven Development

Markus Aleksy, Tobias Hildenbrand, Claudia Obergfell, Martin Schader, Michael Schwind
University of Mannheim, Schloss, D-68131 Mannheim, Germany
{aleksy, hildenbrand, schader, schwind}@uni-mannheim.de, claudia.obergfell@arcor.de

Abstract:
A common problem in model-driven software development (MDSD) processes is the tracing of requirements across different phases of the software development life cycle and multiple levels of abstraction down to the code level. Because debugging at the model level is not feasible yet, unwanted or unexpected behavior of the executable system needs to be analyzed at the code level at run-time and in a feedback loop must be traced back to and handled at the model level. Thus, traceability is a very important success factor and quality criterion in software engineering and maintenance and especially when developing high-quality model-driven infrastructures. In this paper, we present the conceptual design and prototypical implementation of a lightweight traceability approach which supports tracing requirements across different models and levels of abstraction. While providing support for representing different types of traceability links between design models and implementation details, our approach can easily be integrated into existing MDSD projects without increasing their complexity.

1 Introduction

Model-driven software development aims at raising the level of abstraction of development processes by describing software systems using formal models on different levels of abstraction, which are ultimately used as a basis for automatic code generation (cf. [31], [12]). Ideally, all changes to an existing system are applied to the model level and propagated to the code level by performing transformations and code generation procedures. Software development efforts benefit from this approach in several ways: according to Mellor and Balcer [34], Bézivin [10] as well as Booch et al. [11], the two main goals of MDSD are to improve the robustness of software artifacts to changes applied to a software system [9] and, more important in this context, to increase the level of abstraction, allowing to better deal with the problem of complexity. It is argued that any approach addressing these problems will ultimately result in a reduction of cost and time to market, which are the main selling arguments given by MDSD advocates. However, the optimistic outlook of the MDSD proponents [35] is not shared by all experts in the software engineering discipline (e.g., [25]).

Despite automatic model transformations, establishing traceability is not a trivial in MDSD projects. Due to the large number of artifacts and their interdependecies, this task is time

consuming, tedious and error-prone [20]. A few basic preconditions for effective and efficient traceability management have been identified in theory and practice, most importantly:

- *automatic recovery, validation, and update* of traceability links [5] and
- *tool integration* (in particular with respect to widely used and possibly heterogeneous development environments) [16].

Therefore, we propose an approach that enables and supports traceability in a model-driven context by allowing the creation and management of three types of explicit traceability links: (1) between requirements and model elements, (2) between model elements at different levels of abstraction, and (3) between model elements and code sections. Based on these three types of *explicit* traceability links, *implicit* links between requirements and code sections realizing them can be derived—and thus horizontal and bi-directional traceability, as required by many industry standards (e.g., [14]), can be achieved even in complex distributed development projects. For instance, this traceability information can be exploited to verify that all specified requirements are reified in code.

The approach we suggest has been designed to satisfy both of the previously mentioned requirements, i.e., automation and tool integration. Due to the fact that many publications neglect the importance of tool support for end-to-end traceability (cf. section 6), we have chosen a pragmatic approach, that can be integrated with existing MDSD tool chains and is suitable to interoperate with widely accepted collaboration platforms. Following a fundamental design science research methodology [26], we have implemented a prototypical tool support and integration with existing development environments. The software prototype is used to demonstrate that our approach is applicable in distributed settings, can easily be integrated (in our case with the Eclipse and CodeBeamer development platforms [17, 29]), and can be applied in practice. The prototype also shows that our approach can be automated and does not necessarily require extensive and time-consuming manual trace capturing. To evaluate the novelty and utility of our approach we have conducted an informed comparison with related research (see [26, pp. 85]).

The remainder of this paper is structured as follows: after a fundamental discussion on the role of traceability, in particular in the context of MDSD (section 2), the conceptual design of our prototype is presented in section 3. The main part of this paper (cf. section 4) contains a description of the prototype that has been implemented to demonstrate the applicability of the traceability concept described. This is followed by a discussion of related research efforts as compared to our approach. The paper concludes with a summary of our findings and an outlook on future developments.

2 The Role of Traceability in Model-driven Development

MDSD approaches rely on two basic assumptions: first, that all requirements to a system are fully and precisely reflected by the models, and second, that each model element

is ultimately transformed accurately into executable application code. The first requirement is relaxed by some interpretations of MDSD, such as architecture-centric MDSD approaches [39], that rely on manual completion of the generated code and concepts, i.e., by defining protected code regions. Common to all MDSD approaches is the raised level of abstraction and the problem of tracing requirements and other artifacts from the model level to the source code level, sometimes across several intermediate model representations [1]. To support traceability in MDSD, it is necessary that the relation between each requirement, its representation in the models (e.g., as model elements, such as classes or associations) and the resulting code sections can be captured, managed, and analyzed [16]. Thus, traceability is generally a critical success factor and quality criterion in model-driven development projects.

Model transformation is one of the key features of MDSD: abstract models are transformed into more and more concrete models and ultimately into source code. In the terminology of the Model-Driven Architecture (MDA), a Platform-Independent Model (PIM) is successively transformed into a Platform-Specific Model (PSM), which is then transformed into code. While the idea of a stepwise reduction of the level of abstraction through repeated model transformations is an integral part of the OMG's MDA initiative, other approaches pursue a model-to-code transformation-based strategy, where either no intermediate models are used or where they are not intended to be manipulated by the modeler.

Either way, traceability plays a central role throughout the MDSD process, because it enables developers to maintain an insight on the relationships between various artifacts, on different levels of abstraction, and, in the case of multiple successive transformations, even between individual transformations. For example, traceability between elements of one model can help to identify both dependencies and commonalities (e.g., two model elements based on a given requirement, cf. [1]). In locally distributed MDSD projects, where people at various sites collaborate on different parts of the models, establishing and maintaining traceability becomes even more difficult [37]. When traceability relations are managed correctly and efficiently, however, traceability can substantially support coordination in both distributed and collocated software projects [36].

As already mentioned before, traceability between requirements and their representation in the models is crucial to ensure that the relevant set of requirements is accurately elicited and eventually implemented in the code. Additionally, traceability information constitutes a suitable basis for further transformations and code generation [33]. Moreover, traceability between a model element and the code generated from this element is important for debugging generated code and for program comprehension in general. While syntactic correctness of both input and output artifacts, i.e., models, source code files, or XML configuration data, as well as adherence to a particular metamodel can be verified, it is difficult to trace unexpected behavior of the resulting system back to one of the more abstract model levels.

Finally, traceability between individual transformations is a necessary pre-condition in order to decompose complex model transformations into modular steps. An example can be found in [40], where information about preceding transformations is used in order to determine which transformation steps should be performed later on in the process. MDSD development processes are well-suited to be (locally) distributed, either by partitioning

application development into sub-tasks, or by assigning application and infrastructure development tasks to different teams (cf. [39]).

3 The TRACES Approach

In this section we provide an overview on the concepts our approach is based on and the different kinds of traceability links that are supported.

3.1 Assumptions

We assume that requirements are represented by a textual description (free text) and a unique identifier. This notion also includes requirements captured in collaboration tools, such as CodeBeamer [29]. We also assume that each model element, such as a class or an association, has a unique identifier. This way, requirements and model elements can be referenced unambiguously. No assumptions about the format of these identifiers are made, however, as long as their uniqueness is guaranteed.

Since the field of model-to-model transformations is still an emerging one, our initial prototype is based on a development process where code is directly generated from the models. Nevertheless, these models are both platform-independent and technology-independent. As a basis for our prototype we have used the OMEGA modeling and code generation infrastructure [22, 23], which will be described briefly in section 4 in conjunction with CodeBeamer.

With regard to code generation, it is assumed that the source code is generated in its entirety, i.e., no manual completion of the generated source code is required. As a consequence, each individual code section is based on one ore more specific model elements. Additionally, all relevant model elements, i.e., classes, associations, etc., including their identifiers need to be available at generation time.

3.2 Explicit Traceability Links

Based on the assumptions described in the previous section, traceability between requirements and model elements is achieved by creating detailed design models where each model element references all requirements it represents. Since all requirements have unique identifiers (cf. section 3.1), these references are unambiguous.

Traceability between model elements and code sections is achieved by creating appropriate references during code generation.These references can be introduced into the code and can contain the identifier of the model element a section is based on. That way, unambiguous references to model elements are created in the code. These can be exploited to trace run-time problems back to specific elements in the original model, which means

that changes necessary to solve these problems can be applied at the model level and, thus, debugging in MDSD environments is facilitated.

The creation of explicit traceability links is the responsibility of the modeler, and to some degree of the MDSD infrastructure provider. While the former is responsible for introducing references from requirements to model elements manually when modeling a system, the latter must make sure that the resources used in the MDSD process, such as source code templates and code generators, are prepared to propagate these references across abstraction levels and to introduce them into the generated output. Explicit traceability links represent the developers knowledge about requirements and their realization in the developed system. They consist of references between model elements and requirements descriptions that cannot be introduced automatically. Additionally the introduction of traceability links requires an object model that contains the information on how a particular traceability link is represented throughout the model transformation and code generation process. While the former must be created for each individual model, the latter is part of a reusable infrastructure and only needs to be created once.

3.3 Implicit Traceability Links

In addition to the retrieval of explicit traceability information as described in the previous section, implicit traceability links between requirements and code sections can be automatically derived using explicit links (cf. requirements in section 1). The creation of implicit traceability links can help to gain a more complete insight into a system by showing relationships between artifacts that the developer may not have been aware of.

For example, from the knowledge that model element A is involved in realizing requirements R_1 and R_2 and that code section C is based on model element A, we can conclude that code section C is involved in realizing requirements R_1 and R_2, as well. Thus, we can use the available information, for instance, to check if all requirements are implemented in the application code. Similarly, the knowledge that any two model elements reference the same requirement could be used to derive a traceability relationship between these two model elements.

The creation of implicit or inferred (cf. [1]) traceability links requires an automatic analysis of existing explicit links in order to identify relationships that the developer has not explicitly modeled, but that are of importance, e.g., when performing change impact analyses to existing systems. Our approach supports this kind of analysis because each requirement and each model element is given a unique identifier (cf. section 3.1). Based on these identifiers, implicit links between requirements and code sections and between model elements can be retrieved by parsing models and code.

4 OMEGA TRACES Prototype

We have implemented a prototype to demonstrate the utility and the practical applicability of our approach. The prototype is fully integrated into the OMEGA modeling and code generation infrastructure. Due to the fact that our approach does not rely on being used in conjunction with OMEGA, but can be used with other MDSD tool chains, only a short introduction is provided. For a more comprehensive discussion, the reader is referred to [22] and [24].

4.1 The OMEGA Approach and Architecture

OMEGA (Ontological Metamodel Extension For Generative Architectures) is an approach to model-driven development that is targeted at facilitating the rapid development of domain-specific modeling and code generation tools. The approach draws from Executable UML (cf. [34]), i.e., it uses class and state chart models to describe software systems at an abstract level. It strongly promotes the reuse of code generation artifacts, such as model transformation scripts and source code templates. Instead of using a general-purpose modeling language, OMEGA relies on domain-specific languages, represented using hierarchies of domain metamodels, e.g., for the domain of web applications. The use of metamodel hierarchies has been suggested by Atkinson and Kühne [9]. OMEGA draws from this idea of describing problem domains on various levels of abstraction.

Based on the theoretical concepts outlined here, a prototype has been implemented as an extension to the Eclipse development environment. Even though OMEGA relies on some assumptions that clearly distinguish it from other model-driven software development approaches, our traceability framework does not depend on these assumptions or any other specifics of OMEGA. Therefore, its use is not restricted to the OMEGA environment. Rather, it can be adapted to be utilized in other code generation environments as well (cf. [22]).

The current OMEGA prototype consists of three Eclipse plugins; the core is a generic modeling tool which is supplemented by a metamodel implementation and a simple, template-based code generator. Figure 1 shows an overview of the components that constitute the OMEGA infrastructure; a detailed description can be found in [22].

Based on (1) the OMEGA modeling environment (section 4.2), the following subsections relate the requirements outlined in the previous section to existing tool support, also for ensuing process steps of (2) code generation (section 4.3), (3) explicit and implicit trace capturing and validation (section 4.4) as well as (4) traceability information management and visualization (section 4.5).

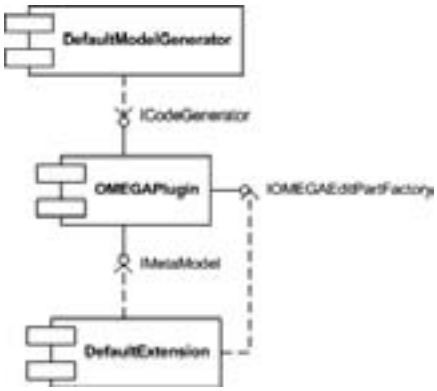

Figure 1: Basic structure of the prototype as described in ([22], p. 142)

4.2 Modeling

The modeling tool consists of a model editor and a number of views which allow a user to edit models viewing specific aspects of them. It supports both static (class diagrams) and dynamic (state chart) models. Additionally, easy to use requirements management, allowing a user to add, edit, or delete project-specific requirements using a graphical user interface (GUI) is supported.

To help automating traceability management, the software handles identifiers for requirements and model elements automatically: whenever a new requirement or model element is created, a unique identifier for that artifact is automatically created. All identifiers are displayed in the GUI. The identifiers of requirements are used when registering or unregistering references to an arbitrary selection of requirements to a model element—which can also be accomplished using the GUI. These references are then stored with the model element in the serialized model.

Models are written to files in a special XML format using the XStream [13] library or, alternatively to XMI, using the Eclipse ECore API ([18]). This way, all model data including model elements as well as references to requirements and information on potential submodels are stored in a single XML file, allowing external tools to retrieve traceability information by parsing the model files.

4.3 Code Generation

The code generator takes a model from the modeling tool as input and transforms it to a format more suitable for code generation than the original one. Based on this "generator model", application code is generated automatically, using a template-based approach [15]. During this transformation, all references to requirements are preserved.

For the purpose of generating executable systems from the generator model, the Velocity Template Engine [8] is used. The template engine has access to all model data using

a context object, which also includes the identifiers of the model elements. Thus, the inclusion of appropriate statements in the templates allows the creation of comments in the source code referencing the model elements each code section is based on.

4.4 Trace Capturing

When using the TRACES tool for capturing traces among requirements and models, the two types of traceability links described in section 3 need to be distinguished. In the following sections, we describe the ways in which *explicit* and *implicit* traceability links are handled in our prototype.

4.4.1 Explicit Traceability Links

Traceability links between requirements and model elements are created using the modeling tool. This is done by adding references to all relevant requirements to the model elements. Figure 2 shows a screenshot of the user interface for associating textually represented requirements with model elements in the development environment. The attribute `DateOfBirth`, selected in the lower section of the screen is needed to realize the requirement of storing the date of birth of every customer.

The relationship between model elements and requirements usually is a many-to-many relationship, meaning that several model elements can reference the same requirement, and that one requirement can be realized by one or more model elements. OMEGA supports this kind of relationship by allowing references to an arbitrary number of requirements for each model element. As figure 2 shows, the list of requirements uses check boxes allowing a modeler to select and add references to an arbitrary number of requirements.

Traceability links between model elements and code sections are created automatically during the code generation process. The associations between model elements and output artifacts are preserved in all internally used intermediate model representations and thus can be introduced into the generated output resources, such as Java source code files. No additional user input is necessary to do this, as the generator model contains all the information needed and the format of the links is defined by the generator templates. While this procedure increases ease of use, it also has a disadvantage: the templates defined for the output artifacts must contain variables that the code generator can replace with references to the model level. Therefore, an initial adjustment of the templates to being used in conjunction with the traceability module is required.

4.4.2 Implicit Traceability Links

OMEGA TRACES supports the extraction of implicit traceability links as described in section 3.3 by serializing the models in a special XML format, which also contains the explicit links between model elements and requirements.

Currently, the prototype relies on external tools for the extraction of implicit traceability

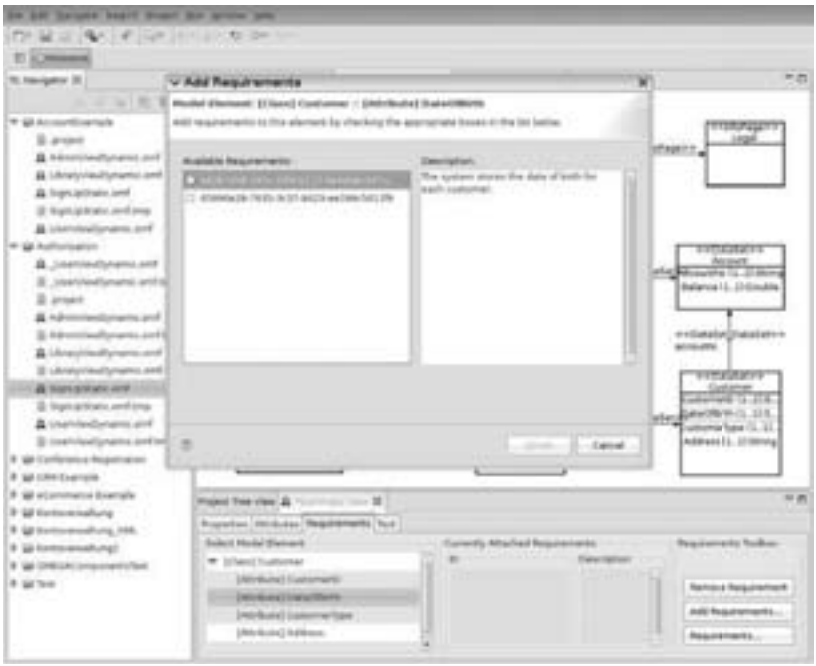

Figure 2: Creating traces between model elements and requirements

links from serialized models and source code artifacts. We have used our prototype in conjunction with the TraVis trace management and visualization tool (cf. [27]) and in the following sections will provide an overview on our findings.

4.4.3 Validation of Traceability Links

Our approach supports validation of traceability information in various ways. On a most basic level, links between model elements and related requirements can be validated automatically. Thus, using each requirement's unique identifier, requirements that no longer exist can be identified and references to these requirements can be deleted. Due to performance issues, a lazy validation and update policy was chosen for this kind of trace validation in our prototype. Therefore, references to requirements are validated each time a model element is accessed. Invalid references are then deleted automatically.

The correctness of links between model elements and related requirements has to be validated manually, however, since the modeller ist responsible for introducing references from requirements to model elements (cf. section 3.2). Consequently, the modeler also must make sure that the references that have been introduced remain valid. To facilitate the discovery of broken traceability links, trace visualization and management facilities can be employed as described in the following section.

4.5 Trace Management and Visualization

In order to support model-driven development efforts in locally distributed scenarios, we suggest the use of a *collaborative software development platform* (CSDP), that supports the management of all artifacts of a development project and provides controlled access to these resources to the stakeholders involved.

Additionally, the CSDP contains different synchronous and asynchronous communication means as well as an embedded wiki engine to facilitate collaborative documentation and information sharing. In our approach, requirements, in the form of issue tracker items, models of different levels of abstraction as well as the source code of an application are managed within a CSDP. Every artifact is supplied with a unique identifier, which also enables the built-in wiki engine to refer to these, e.g., when writing the documentation for future maintenance (see [27] and [29] for concrete features of the CodeBeamer CSDP).

Thus, the associations among elements can be managed as CSDP hyperlinks. Moreover, we have developed a complementary tool for trace visualization (TraVis) that is tailored at supporting traceability management even in distributed MDSD settings (see figure 3). Commonly used requirements management tools, on the other hand, do not provide full end-to-end traceability, whereas commonly available collaboration platforms lack integrated support for collaborative requirements elicitation, specification, and management processes (cf. [27] for a more detailed demarcation of existing traceability management tools).

Figure 3: Visualizing traces between models, requirements, and tracker items

Traces among requirements and different models can be extracted from the CSDP and project managers or developers are enabled to manage the traceability information visually, i.e., discover and repair inconsistencies. In addition, TraVis provides functionality for

conducting impact analyses starting from one particular artifact, e.g., a requirement or model, and exploring those artifacts affected when changing something at the point of origin. This in turn allows for detailed calculations on the estimated entailing costs of a particular change and thus its economic feasibility [37].

5 Contributions and Limitations

In the previous sections we have described the set of tools that support our approach to tracing requirements across different stages of artifact elaboration in model-driven development processes. The main contribution of our approach consists in revealing and proposing a prototypical solution for maintaining end-to-end traceability in distributed model-driven development processes. In doing so, a continuous and integrated tool chain for supporting software modeling and code generation (OMEGA), trace capturing and validation (CodeBeamer), well as trace information management and visualization (TraVis) is presented.

Moreover, the basic applicability of the approach has been shown by implementing the tool chain mentioned above based on both existing development tools and integrated custom-built prototypes that have already been evaluated in other related contexts (see [27]).

Future extensions to the prototype will include tools for parsing models and code. These will provide the means to perform a number of tasks, such as automatically deriving implicit traceability links from the explicit links (cf. section 3.3), the validation of these links, and the interpretation and visualization of the traceability information. TraVis is currently evaluated and further adapted to the TRACES approach for visualization of traces among model elements stored in a CSDP [37].

6 Discussion and Related Work

The general use of identifiers and references is a basic technique utilized in a wide range of application domains, so it will not be discussed further, here. In this section, we will provide an overview of related research efforts in the field of requirements traceability in order to justify the TRACES approach and highlight its unique features as opposed to existing solutions. This way, an evaluation of the approach's unique utility is conducted descriptively and based on the current body of knowledge in software engineering and information systems research [26].

There are many instances in literature describing automatic or semi-automatic approaches to trace capturing and management. For example, Antoniol et al. have developed a number of approaches based on information retrieval (IR) techniques [4, 2, 6, 3] and on the observation of test scenarios [7] that are largely supported by tools automating the process. Similar research on applying IR for candidate link generation has been conducted by Huffman-Hayes et al. [28]. However, these approaches inherently do neither support

MDSD processes nor distributed collaboration processes. TRACES, on the other hand, was purposefully designed to support this class of software engineering problems.

Egyed also suggests the use of test scenarios in order to obtain traceability information [19, 20, 21]. This approach can only be partly automated, though, due to the fact that traces need to be established and managed mainly manually. Spanoudakis et al., on the other hand, describe a rule-based approach which can almost completely be automated [38]. So do Jirapanthong and Zisman, who develop another rule-based approach to the automatic generation of traceability links [30]. Since model-to-model and model-to-code transformations also contain rules, these approaches are somewhat comparable to ours. However, as it is the case with the IR approaches, none of the rule-based approaches can be smoothly integrated with existing MDSD and CSDP tools in order to support distributed MDSD.

There are only a few authors emphasizing the importance of tool integration with respect to traceability. Even though some tool support for traceability, i.e., trace capturing, management, and analysis exists, Aizenbud-Reshef et al. conclude that there is a lack of integrated solutions [1]. Complementarily, Kowalczykiewicz and Weiss stress the importance of tool integration [32]. They present a first prototype for an integrated tool platform with support for traceability and change management whereas there are no particular capabilities for MDSD yet.

Therefore, the unique utility of the OMEGA TRACES approach consists of allowing a relatively high degree of automation with respect to trace generation and capturing traces among requirements and model elements at different levels of abstraction due to a broad information basis stored centrally within the collaboration platform. Moreover, TRACES is designed to integrate with existing development environments and collaboration platforms in order to support collaboration in distributed model-driven development processes commonly found in outsourced and offshore software projects (cp. [37]). In addition, the integration with TraVis accounts for visual traceability and change management in distributed projects (see section 4.5). As has been shown, both analytically and empirically, in [27], collaborative and visual traceability management accounts for superior productivity as compared to commonly available solutions. Therefore, this custom-developed solution is chosen here in combination with the TRACES prototype.

7 Conclusion

We have presented an easy to use, yet inherently powerful approach to traceability and a prototypical implementation, which allows a high level of automation, tool integration, and distributed collaboration. By integrating our prototype with the Eclipse and CodeBeamer platforms, we have demonstrated that the approach can easily be used in conjunction with widely accepted development environments for realizing locally distributed MDSD projects. Moreover, our prototype supports automated generation, validation, and update of traceability information.

A main advantage of our approach is the ease of use and therefore easy adoption for devel-

opers, since little additional effort is necessary to create a reasonable amount of traceability information that can be retrieved from models and code using external tools, such as TraVis. Due to the fact that all information needed to link requirements to model elements and source code and other textual resources is stored in XML/XMI-serialized models, our approach can easily be adapted to other modeling and code generation environments that use these standards.

There are, however, a few minor issues that will be addressed in the near future. Most significantly, the current strategy of validating and updating traceability links allows "invalid" model states where model elements reference requirements that no longer exist. This is because a lazy validation and update policy was chosen for the prototype to improve its performance. However, a more efficient solution for this particular problem will be developed and evaluated empirically in case study and/or experimental settings.

References

[1] Aizenbud-Reshef, N., Nolan, B. T., Rubin, J. and Shaham-Gafni, Y.: Model Traceability. IBM Systems Journal, **45** (2006) 515-526

[2] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A. and Merlo, E.: Tracing Object-Oriented Code into Functional Requirements. Proceedings of the IEEE International Workshop on Program Comprehension (IWPC), Limerick (June 2000)

[3] Antoniol, G., Canfora, G., Casazza, G. and De Lucia, A.: Information Retrieval Models for Recovering Traceability Links between Code and Documentation. Proceedings of the IEEE International Conference on Software Maintenance (ICSM), San Jose (October 2000)

[4] Antoniol, G., Canfora, G., De Lucia, A. and Merlo, E.: Recovering Code to Documentation Links in OO Systems. Proceedings of the IEEE Working Conference on Reverse Engineering (WCRE), Atlanta, Georgia (October 1999)

[5] Antoniol, G., Caprile, B., Potrich, A., and Tonella, P.: Design-code Traceability Recovery: Selecting the Basic Linkage Properties. Science of Computer Programming, **40** 2001, 213-234

[6] Antoniol, G., Casazza, C. and Cimitile, A. Traceability Recovery by Modeling Programmer Behavior. Proceedings of the IEEE Working Conference on Reverse Engineering (WCRE), Brisbane (November 2000)

[7] Antoniol, G., Merlo, E., Gueheneuc, Y.-G. and Sahraoui, H.: On Feature Traceability in Object Oriented Programs. Proceedings of the International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE), Long Beach (November 2005)

[8] Apache Software Foundation: The Apache Velocity Project. `http://velocity.apache.org` (2007)

[9] Atkinson, C., Kühne, T.: The Role of Metamodeling in Model-Driven Development. International Workshop in Software Model Engineering (held in conjunction with UML '02), Dresden (October 2002)

[10] Bézivin, J.: MDA: From Hype to Hope, and Reality. Invited Talk at UML 2003, San Francisco (October 2003)

[11] Booch, G., Brown, A., Iyengar, S., Selic, B.: An MDA Manifesto. MDA Journal, 2004

[12] Brown, A.: Model driven architecture: Principles and practice, Software and Systems Modeling, Volume 3, Number 4 (December 2004), pp. 314-327, Springer, Berlin, Heidelberg, 2004

[13] Codehaus: XStream. http://xstream.codehaus.org(2007)

[14] CMMI Product Team: CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1 Carnegie Mellon Software Engineering Institute, 2002

[15] Czarnecki, K. and Helsen, S.: Classification of Model Trans-formation Approaches. 2nd OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, CA, 2003.

[16] Domges, R. and Pohl, K.: Adapting Traceability Environments to Project-Specific Needs Communications of the ACM, **41** (1998) 12, pp. 54-62

[17] Eclipse Foundation: Eclipse Development Platform. http://www.eclipse.org (2007)

[18] Eclipse Foundation: Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/ (2007)

[19] Egyed, A.: A Scenario-Driven Approach to Traceability. Proceedings of the International Conference on Software Engineering (ICSE), Toronto (May 2001)

[20] Egyed, A.: A Scenario-Driven Approach to Trace Dependency Analysis. IEEE Transactions on Software Engineering (TSE), **29** (2003) 2, pp. 116–132

[21] Egyed, A.: Resolving Uncertainties during Trace Analysis. Proceedings of the 12th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), Irvine, CA (November 2004)

[22] Gitzel, R.: Model-Driven Software Development Using a Metamodel Based Extension Mechanism for UML. Peter Lang Europäischer Verlag der Wissenschaften, Frankfurt (2006)

[23] Gitzel, R., Ott, I., Schader, M.: Ontological Extension to the MOF Metamodel as a Basis for Code Generation, in: The Computer Journal, 50 (2007) 1, pp. 93–115

[24] Gitzel, R., Schwind, M.: Using Non-linear Metamodel Hierarchies for the Rapid Development of Domain-Specific MDD Tools. Proceedings of Software Engineering Applications (SEA), Dallas, Texas (November 2006)

[25] Glass, R.L.: The Generalization of an Application Domain. IEEE Software, **17** (2000) 5, pp. 127–128

[26] Hevner, A. R., March, S. T., Park, J., Ram, S.: Design Science Information Systems Research MIS Quarterly, MISQ Discovery, **28** (2004) 1, pp. 75–105

[27] Hildenbrand, T.: Improving Traceability in Distributed Collaborative Software Development – A Design Science Approach. Dissertation Thesis, University of Mannheim, Germany (2008)

[28] Huffman Hayes, J., Dekhtyar, A., Sundaram, S. K.: Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. IEEE Transactions on Software Engineering, 32 (2006) 1, pp. 4–19

[29] Intland Software: Collaborative Software Development Solution. http://www.intland.com/products/codebeamer.html (2007)

[30] Jirapanthong, W. and Zisman, A.: Supporting Product Line Development through Traceability. Proceedings of the Asia-Pacific Software Engineering Conference (APSEC), Taipei (December 2005)

[31] Kleppe, A., Warmer, J. and Bast, W.: MDA Explained - The Model Driven Architecture : Practice and Promise. Addison-Wesley, 2003.

[32] Kowalczykiewicz, K. and Weiss, D.: Traceability: Taming uncontrolled change in software development. Proceedings of the National Conference on Software Engineering, Tarnowo Podgórne, Poland (2002)

[33] Lindvall, M., Sandahl, K.: Practical Implications of Traceability. Software – Practice & Expererience, **26** (1996) 10, pp. 1161-1180

[34] Mellor, S., Balcer, M.: Executable UML - A Foundation for Model-Driven Architecture. Addison-Wesley, Hoboken, 2000.

[35] Miller, J., Mukerji, J.: MDA Guide Version 1.0.1, OMG Document Number: omg/2003-06-01, OMG, `http://www.omg.org/cgibin/doc?omg/2003-06-01`

[36] Redmiles, D., van der Hoek, A., Al-Ani, B., Hildenbrand, T., Quirk, S., Sarma, A., Filho, R. S. S., de Souza, C.R.B. and Trainer, E.: Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects WIRTSCHAFTSINFORMATIK, 49 (2007) Special Issue, pp. S28-S38

[37] de Souza, C.R.B., Hildenbrand, T. and Redmiles, D.: Towards Visualization and Analysis of Traceability Relationships in Distributed and Offshore Software Development Projects. Proceedings of the First International Conference on Software Engineering Approaches For Offshore and Outsourced Development (SEAFOOD'07), Springer Lecture Notes in Computer Science (LNCS), Zurich, Switzerland (February 2007).

[38] Spanoudakis, G., Zisman, A., Perez-Minana, E. and Krause, P.: Rule-based generation of requirements traceability relations. Journal of Systems and Software, **72** (2004) 2, pp. 105–127

[39] Stahl, T., Voelter, M.: Model-Driven Development: Technology, Engineering, Development. Wiley, 2006

[40] Vanhooff, B. and Berbers, Y.: Breaking Up the Transformation Chain. 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05), Workshop on Best Practices for Model Driven Software Development, San Diego (October 2005)

On the Role of Communication, Documentation and Experience during Testing - An Exploratory Study

Timea Illes-Seifert, Barbara Paech
University of Heidelberg,
Institute of Computer Science, Im Neuenheimer Feld 326
Germany-69120 Heidelberg
{illes-seifert, paech}@informatik.uni-heidelberg.de

Abstract: Nowadays, the quality of software is becoming more and more a competitive factor. As complete testing is impossible, testers have to make decisions, e.g. to choose which parts of the software have to be tested in which way. For this purpose, testers need a lot of information, such as input documentation which serves as a basis for the derivation of test cases or information on the project status which serves as a basis for planning the testing process. Thus, testers rely on up-to-date and complete information in order to make sound decisions. Consequently, the quality of the testing process depends on the quality of the information sources available for the testers. This paper presents the results of an exploratory study conducted during the SIKOSA research project with expert testers of our industry partners in order to identify the most valuable sources of information during testing. Particularly, we conducted interviews in order to investigate which documents are often used by testers, as well as the role of communication and experience. Our results show that defect reports are very valuable. User manuals and problem reports are equally important, because they represent real usage of the software and serve testers as an input for realistic test cases. In addition, our results show the influence of an independent testing team on test process characteristics.

1. Introduction

Spectacular software failures like the crash of the Ariane 5 rocket [Do97], but also software failures which occur daily in business software show that testing activities are essential in order to detect defects before release. However, complete testing is impossible, and as a consequence, testers have to make a lot of decisions during the testing process in order to constrain the set of potentially infinite number of test cases to a set which can possibly detect the most critical defects. Thus, testers make decisions, e.g. on the test design technique to be used in order to derive test cases or on the test data which serve as input for test cases. In order to conduct all these decisions thoroughly, testers need complete and up-to-date information, e.g. about requirements, project status, etc. The main assumption of our work is that *the better the information and the information flow between testers and other project members (e.g. requirements engineers or project manager) is, the better will be the quality of the decisions made during testing*. The knowledge of testers' information needs allows to provide testers with the right information at the right time and to define the best way of providing it (e.g. documented, verbal). Based on this knowledge, test process improvements can be designed and implemented.

In this paper, we present the results of an exploratory study, performed during the SIKOSA research project with expert testers, the aim of which is to analyze information flow within the testing processes. Particularly, we analyze which documents are frequently used and which roles are consulted when making decisions during testing. In addition, we investigate the role of experience needed to make sound decisions. The results of the study serve as a basis for recommendations regarding the optimization of a testing process.

The remainder of this paper is organized as follows. Section 2 introduces a conceptual decision framework, containing decisions to be made during the testing process. We used this framework as the basis for our study during data collection and analysis. In Section 3, related work is presented. Section 4 describes the design of the study. We conduct expert interviews for exploring information flow patterns in testing. Section 5 presents the findings whereas Section 6 deals with the threats to validity of our study. Finally, Section 7 concludes the paper.

2. Decisions within the Testing Process

This section introduces some basic concepts and the decision framework which served as a basis for the exploratory study. In previous research work, we analyzed the testing process [IHPR05], [BIL07a], [BIL07b] and developed a decision framework, which identifies the decisions to be made during the testing process and assigns them to decision levels.

Test planning and control (TP&C). Since testing is a complex process, thorough planning and monitoring is needed. Consequently, during TP&C activities testers decide on schedules, resources, and efforts estimated for testing activities, as well as on risks (which threaten the successful completion of the testing).

Test strategy definition (TSD). The main goal of TSD is to define which parts of the software should be tested in which way (e.g. how intensively or with which test design techniques) in order to find the most critical defects fast. Correspondingly, testers decide on test end criteria, defining conditions which have to be fulfilled to finish testing activities. In addition, decisions on the test design techniques used to develop test cases (to find the most critical defects) have to be made and a test model can be selected. A test model, e.g. a state model facilitates the derivation of test cases. Closely related to the selected model, the decision on the representation for the model, as well as coverage criteria, e.g. transition coverage or state coverage in case of a state model can also be decided during TSD. In the case of automation, testers also have to decide on the degree of automation.

Test analysis and design (TA&D). During test analysis and design activities, testers decide on test cases including test steps, test data and test sequences. In addition, testers review the documentation to be used as input for testing activities and decide on their quality, e.g. testers make decisions about the testability of the requirements specification document. If the requirements specification does not fulfil the required quality from the testers' point of view some rework is needed.

Test execution (TE). During test execution, decisions on the evaluation of the executed test cases (called test runs) have to be made. Consequently, testers have to decide, whether a test run revealed a defect or not.

Test cycle evaluation (TCE). During test cycle evaluation, the results of the test runs have to be analyzed. Thus, testers check if the test end criteria have been fulfilled and decide whether testing activities can be finished or not.

3. Related Work

Similar work, analysing information gathering strategies of maintainers is described in [S202] and in [TL01]. Most related work focuses on the description of the test process, e.g. the fundamental test process presented in [SLS06] addresses phases and activities to be passed through when testing a software system. The IEEE standard for software test documentation [IEEE98] specifies all artefacts to be created during the testing process, e.g. the test plan; the information flow, as well as the information sources needed are not part of the standard. Another group of related work represents test process improvement models like TPI (Test Process Improvement) [KP02] or test maturity assessment models, e.g. TMM (Testing Maturity Model) [BSC96]. The focus of these models is not the information flow within the testing process, but the steps for its improvement, respectively on criteria to assess the maturity of the organizational testing process. None of the presented references contains empirical studies. The work which is most related to our work is [Da05]. The authors present guidelines for requirements engineering practices which facilitate testing. In contrast to the work in [Da05] which addresses requirements engineering processes and artefacts, this study has a larger focus including other information sources of the software development project. In addition, we analyze communication, as well as the role of experience during testing. To our knowledge, this is the first study exploring the information flow during the testing process in detail. Empirical studies are essential in understanding the nature of information processes. This is also the case with the testing process. By this study, previously formulated advices in literature which are not supported by empirical studies could be confirmed. For example, the outstanding role of the requirements specification and of previously found defects for the testing process could be confirmed. This study, however, also allows insights which have not been yet considered in literature, e.g. the role of the user for testers. Knowing that information from customer is so valuable for testers, processes can be adapted appropriately in order to facilitate the information flow from customers to testers.

4. Study Design

In this chapter, we present the research questions; we introduce the characteristics of the participants of the exploratory study and provide an overview of the data collection and data analysis methods used to gather and to investigate the data.

4.1. Research Questions

Subsequently, we will outline the research questions addressed in this study and the rationales behind. The respective questions and rationales are listed in Table 1.

Table 1: Research Questions

Questions	Rationale
Q1: Which documents are frequently used by testers when making which testing decisions?	The main assumption of this research question is that documents are an important information source for all participants of the software engineering process, including testers. To know which documents are frequently used by testers is important, because quality assurance activities concerning information sources often consulted by testers can be intensified purposefully.
Q2: What role does communication play as an information source?	The main assumption of this question is that documentation is never completely sufficient as input to the testing process, so that details have to be clarified in face-to-face discussions. And even if documentation was complete, up-to-date and unambiguous, communication is often preferred to reading documents.
Q3: What is the role of experience in testing?	This is an important question to be analysed, because it is important to know to what extent and for which decisions testers rely on their experience instead of documentation. Knowing this enables to decide: Which activities are suited for test automation? Which decisions are suited to be executed by novice testers?

4.2. Participants

The main criterion for the selection of the participants was their experience in the testing area. As a consequence, all participants out of five organizations had at least three years of experience, and most of the participants had five to ten years of experience. Three participants had even more than ten years of experience. Table 2 summarizes the characteristics of the participants. Organisation A and E develop standard software, whereas the other organisations develop individual software. Only organisation C develops software for in-house use. The testers in organisation A work on the same project, whereas the testers in the organisations C and D work on different projects.

Table 2: Participants' Characteristics

Experience (in years)	Role(s)	Main Tasks	Organisation
>10	Test designer	Test planning Test case design Manual test execution	D
>10	Test designer	Test planning Manual test execution	D
>10	Test manager	Establishment of a standard testing process including supporting tools	B
>10	Tester, Test manager	Test planning Manual test execution	E
10	Test manager, Quality engineer	Test planning Test case design Monitoring system operation	D
10	Test manager, test designer	Test management and control Test case prioritization Human resources management and motivation	A
5	Test manager	Product development, Manual test execution and protocol Coordination of testing activities Product roll-out (= deployment in the productive environment)	C
5	Test designer	Supports test manager in planning activities Test case design Manual test execution and protocol	A
5	Test designer	Test case design Execution of test cases Fault localisation Regression testing	D
3	Test automation engineer	Manual test execution and protocol Test automation: implementation of the test automation framework	A
3	Test manager	Test planning Manual test execution	C

4.3. Study Process

The study was performed as a qualitative study. Qualitative studies use data in form of text, images and sound drawn from observations, interviews and documentary evidence, and analyse it using methods that do not rely on precise measurement to yield their conclusions [Se99]. We used this research method because it helps to gain more experienced with the analysed phenomenon. In our case, our goal was to get a deep understanding of the information flow during the testing processes.

The study was conducted in the form of seven face-to-face interviews and one telephone interview. Three interviewees completed the questionnaire "offline". The interviews were semi-structured, based on a questionnaire sent in advance to the participants. The interviews took three hours on average.

The questionnaire itself consists of three parts. The first part contains questions regarding the testers' experience and role in the organization, as well as questions on the organizational testing process. Particularly, the interviewees were asked about the testing decisions to be made during the testing process in their particular organisation. The second part of the questionnaire addresses communication and documentation sources during testing, whereas the third part contains questions regarding the role of experience within particular activities. In the second part of the questionnaire, the interviewees got a list of documents that could theoretically be used during testing decisions, e.g. requirements specification or design specification. Then we asked the interviewees which documents are needed when making particular decisions. The interviewees were also asked to indicate documents not contained in the list, as well as a "wish list" containing documents currently not available to them. Similarly, we asked the interviewees which specific roles are consulted when making particular decisions. In the third part of the questionnaire, we asked the interviewees to rate the experience needed to make particular decisions. The questionnaire can be found in [IP08].

Data Collection. In the data collection phase, field notes taken during the interviews were coded and stored in a study data base. Coding [Se99] is a procedure which transforms qualitative data into quantitative data by assigning values to qualitative statements. This allows the combination of qualitative and quantitative methods for data analysis. During the coding, interviewees were contacted when ambiguities in the data occurred.

To assure the validity of our results, we used multiple information sources for evidence as recommended in [Yi03]. Thus, beside interviews, document analyses have been performed. We analyzed test case specification templates and test case specifications, test protocols and test process descriptions, as well as input documentation, e.g. requirements in the organization, the testers belong to. Furthermore, we got insight into other information sources like discussion forums. Another aspect considered to assure validity was the representativeness of the interviewees with regard to their qualification, experience and testing tasks. All interviewees are experienced testers, four of them with more than ten years of testing experience.

Data Analysis. For the data analysis, we used different qualitative and quantitative analysis methods. Quantitative methods were used in order to determine patterns and tendencies in the data, e.g. by counting which role is consulted most of all during the testing process. Qualitative methods were used to search for an explanation for these particular tendencies. Thus, we performed cross-case analysis [Se99] and partitioned the data into different categories by using different criteria, e.g. we partitioned the data depending on the testing group's organization as an independent team or not.

5. Findings

In this section, the analysis of the results of the study is presented. First, we detail the test process characteristics, including the roles and decisions mentioned by the interviewees. Then, we discuss the documentation, communication and experience characteristics. Finally, we present the problems during the testing process as mentioned by the interviewees.

5.1. Test Process Characteristics

Test planning and control. With the exception of risk analysis, all decisions to be made during TP&C (as described in section 2) are equally often mentioned to be performed. 9 out of 11 interviewees mention that a particular decision is made during the testing process in their organization. Only about half of the interviewees (6 cases of evidence) cited that a risk analysis is performed when deciding on relevant risks influencing the testing project. Only 4 of the interviewees report that all TP&C related decisions are in the testing team's field of responsibility. Three interviewees even indicate that all TP&C related decisions are performed by persons not belonging to the testing team, mostly by the project manager. In all other cases, TP&C related decisions are partially made by the testing team.

Test strategy definition is a task not well established within the testing processes we analyzed. Only few decisions are indicated to be made, where the definition of test end criteria is a decision mentioned most by the interviewees (9) followed by the selection of the test design technique (5). All other activities are rarely cited.

Test analysis and test design. Decisions on test steps, on test data and on test sequences are indicated to be made by nearly all interviewees, whereas the assessment of the testability, as well as the assessment of the quality of the input documentation is indicated only by about half of the interviewees. These decisions are mostly made by the testing team. Within organizations not having an independent testing team, these decisions are performed by developers (where the "tester" is not the developer of that particular part of the software). Since most organizations do not automate tests, the realization of test cases and consequently all decisions related to test automation are confirmed only by a small part of the interviewees.

Test execution and test cycle evaluation. All interviewees report to make decisions concerning the success or failure of particular test runs. The decision on the test run evaluation is mostly made by testers, in some cases by the whole testing team. The evaluation of a test cycle is only performed by fewer than half of the interviewees.

To sum up, it is not surprising that decisions indicated to be made by almost all interviewees concern test decisions in the narrow sense (test case definition and test execution). However, TP&C, as well as TA&D related decisions are each indicated to be performed on average by 9 out of 11 interviewees. Decisions concerning the TCE, as well as the TSD are made on average by fewer than half of the interviewees. Figure 1 shows the test process characteristics as mentioned by the participants.

5.2. Documentation Characteristics

TP&C related decisions, particularly decisions on effort and schedule, require the most documentation, followed by TA&D decisions, especially decisions on test data and test steps, as well as on the definition of test sequences. The interviewees report a high need of documentation during TCE, especially the requirements and design specification. Decisions during TSD and TE require little documentation.

The role of the requirements specification. The requirements specification is by far the most important document for testers (46% of all decisions need the requirements specification as input, see also Figure 2.). During TP&C, the requirements specification is especially used for decisions concerning effort estimation and scheduling, whereas during TA&D the requirements specification is especially used to decide on test cases (including test steps, test data and test sequences). In addition, the requirements specification is also used during TE in two contexts. First, when testers are pressed for time, they report to use the requirements specification as test specification. In this case, decisions on the test design are made concurrently to the test execution. Second, in case of a failure or of an unexpected behaviour, testers consult the requirements specification in order to analyze if it is a real failure. All testers emphasize the importance of the requirements specification to be up-to-date and complete.

Learning from defects. Previously found defects are a very valuable information source for testers, whereas both defects found by the test team, as well as defects reported by customers are almost equally important (25%, respectively 24% of all decisions require customer problem report respectively bug reports as input, see also Figure 2). Testers report that previously found defects are good indicators for defects in the software because of following reasons:

(1) Many defects persist across different versions. Two categories of persisting defects are reported by testers: *permanent defects*, which occur across all versions and "*jumping" defects,* which regularly "jump" over a constant number of versions.
(2) The correction of a defect introduces more defects.

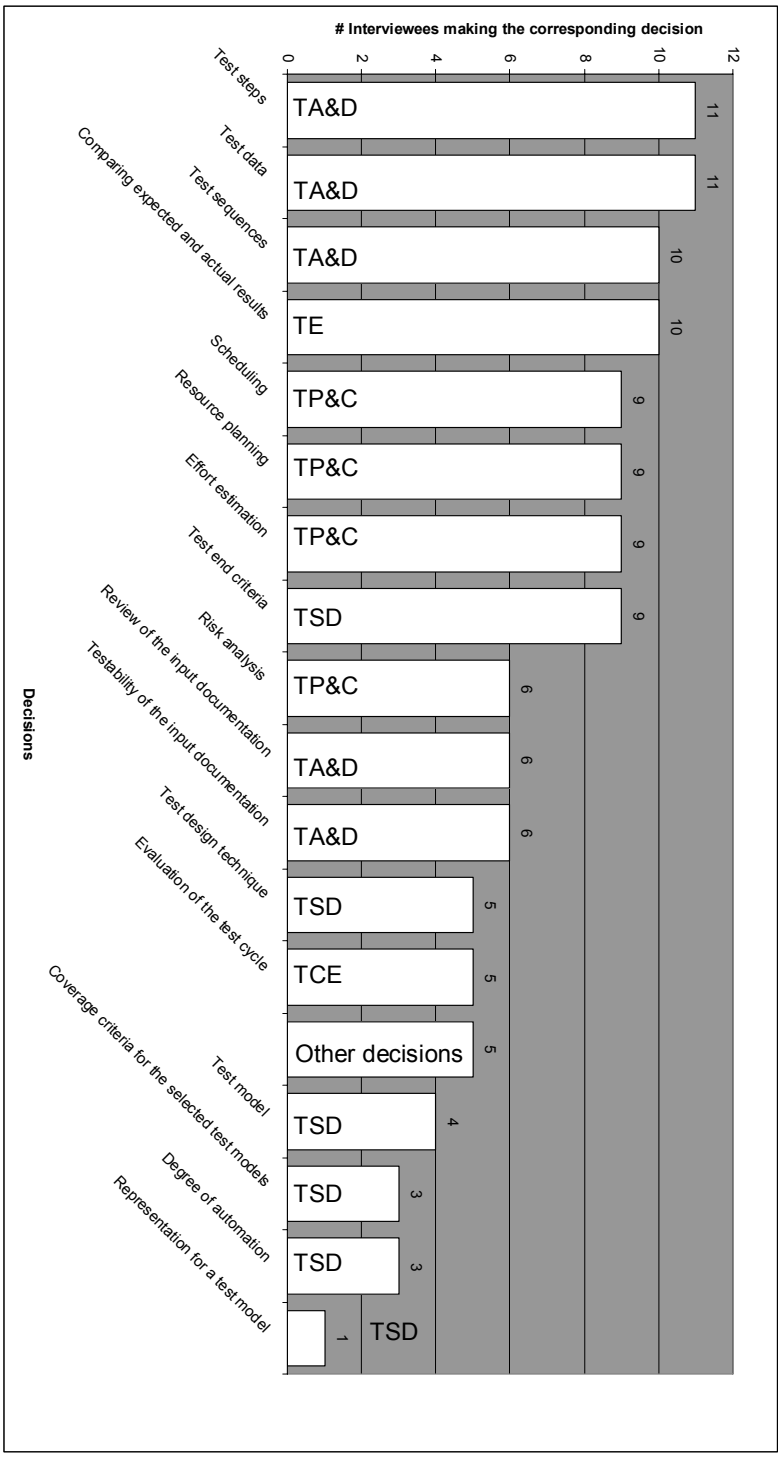

Figure 1: Test Process Characteristics

Knowing potential defects, testers can decide on the test effort to be spent to test particular areas of the software. Defects also serve as input for TA&D. On the one hand, testers select test cases to be re-executed if they revealed a defect. On the other hand, testers develop new test cases on the basis of known defects using different strategies, which we refer to as *intensifying*, *expansion and transferring.*

(1) *Intensifying:* Testers investigate the functionality more intensively and usually vary the test data, for example, or the preconditions of the test case.
(2) *Expansion:* Testers search for functionality used by the functionality which revealed the defect or using this functionality.
(3) *Transferring:* Testers search for similar functionality (which could contain the same defect).

Figure 2 illustrates the documents needed as input during testing as mentioned by the interviewees.

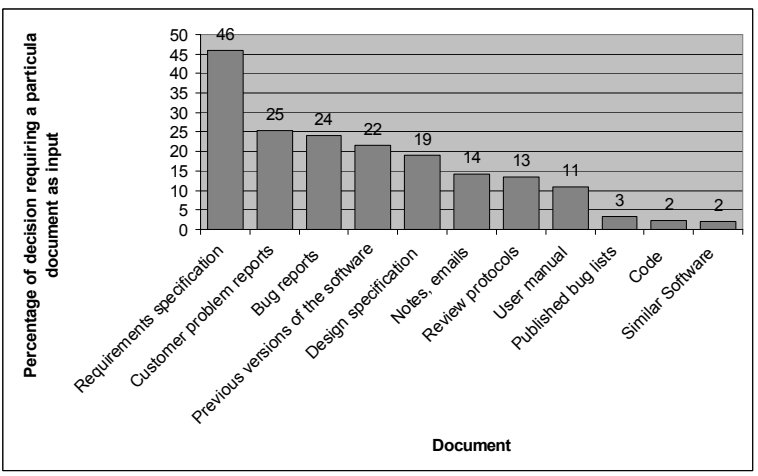

Figure 2: Documentation needs during testing

The role of the user within the testing process. Even though only few of the testers are in direct contact with users, the users play an important role during testing. Using documentation produced for and by users, testers can develop more realistic and more relevant test cases. Thus, testers bridge the gap to the customer by using customer problem reports and user manuals in order to develop realistic test scenarios and in order to define test environments and configurations close to the real productive environments. Consequently, this documentation is very valuable when deciding on test data and test steps. One interviewee also mentioned to use the user manual to get familiar with the software system.

Wish lists. When asked for information sources, which are not available but which were useful for testing, 4 interviewees emphasize that up-to-date and complete requirements are crucial and more important than other documented information sources. Interviewees cite following reasons why requirements specifications are usually not up-to-date and complete: time pressure at creation time, as well as the fact that requirements engineers are not aware of the tester's information needs.

5.3. Communication Characteristics

During the testing process, most communication occurs among the requirements engineer and the project manager followed by the developer. Testers have direct contact with the customer only when the customer is "in-house". Apart from this, there is no direct communication between testers and customers, in spite of their request for this type of communication. Figure 3 shows the percentage of decisions in which a role is involved.

Most communication is reported to take place during TA&D, where the main communication partners mentioned by the interviewees are requirements engineers and project managers. However, during TE, there is also a great need for communication. The main contact persons are requirements engineers and developers mostly in the presence of a failure. Communication during TP&C occurs mostly with the project manager. However, little communication takes place during TSD and TCE.

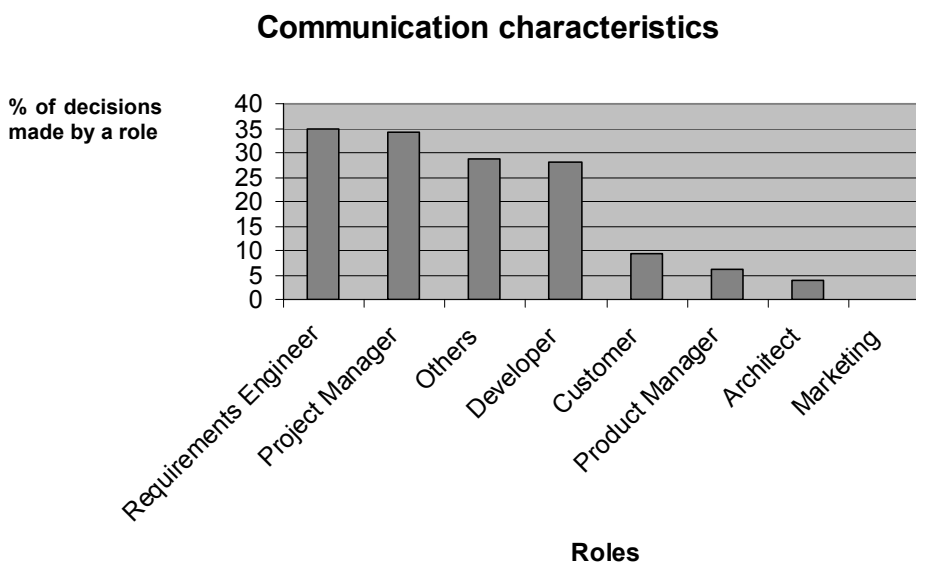

Figure 3: Communication Characteristics

5.4. Experience Characteristics

The interviewees report that most experience is required during TP&C and during TCE. In addition, a lot of experience is required for TA&D. In contrast, little experience is required for decisions related to TSD and to TE.

Among the decisions made during the testing process, the definition of test data is stated to be the decision which requires the system specific experience at most. All interviewees indicate that this decision requires very much experience. In addition, this is the only decision which solely requires system specific experience. Effort estimation and risk analysis, as well as the evaluation of the test cycle are also indicated by the interviewees to require high system specific experience. In general, almost all decisions require more system specific than general experience. Managerial activities, e.g. scheduling, resource planning and effort estimation require the most general experience. As expected, test case execution and evaluation require the least system specific and general experience.

5.5 Problems

In the following, the main problems as mentioned by the interviewees are presented.

Poor quality of the documents used as input, especially poor quality of the requirements specification is one of the major issues during testing. We asked the interviewees to indicate the most severe problems occurring during testing. One of the most frequently mentioned problem concerns the quality of the input documents, particularly the lack of quality of the requirements specification. Only two participants do not indicate poor requirements (e.g. incomplete, unambiguous) as one of the most difficult problems during testing. Three participants especially require more detailed descriptions, particularly concerning pre- and post conditions of a requirement, as well as dependencies between requirements and between the software and its environment (including the software and hardware environment). One of the main reasons for the poor quality of the requirements from the testers' point of view is the lack of involvement in the review process. Only half of the interviewees report that testers are involved in the review process. In one special case, the requirements specification is not reviewed at all.

Testing decisions require system specific experience. Almost all decisions require more system specific than general experience. In addition, testers indicate to rely on their own experience, rather than on experience made by others, as they do not frequently consult published defect lists.

Testers rely on their own experience more than on test design techniques when deciding on test data and test steps. Testers rely more on their own experience than on test design techniques which generate a high amount of test cases and prefer an exploratory-oriented approach. In addition, in the case of time pressure, testers deviate from systematic approaches and reduce the set of test cases according to their own experience. For example, the testers apply equivalence partitioning results in 6 equivalence classes and rate 4 of them as unrealistic and with low potential to detect defects. In this case, they decide to specify and execute only these two test cases which they appraise to be well suited to reveal a defect.

High documentation and communication needs during test execution suggest incomplete descriptions of the expected outcome in test case specifications. Reasons for this are either quality deficiencies in the documentation which served as input for decisions on test cases or shortage of time when testers decided on test cases, leading to incomplete descriptions of the expected outcome.

The results of a test cycle can not be objectively assessed. Surprisingly, testers point out the role of experience in the evaluation a test cycle. One would expect that the evaluation of the test results requires "only" a decision on the efficiency of the test strategy, i.e. "Have the test design techniques been applied and have the test end criteria been met?" But since the test strategy definition is not well established in testing processes, the decisions related to TSD have to be taken later, namely during the test evaluation. In addition, one participant criticizes the lack of a systematic learning process across test cycles.

6. Threats to Validity

One threat to validity of our study is the fact that the results may be specific to the particular interviewees. We addressed this problem by selecting very experienced testers for the interviews. Another threat is the ability to generalize the results due to the fact that we selected a small population. We addressed this problem by using techniques which assure validity of qualitative studies [Se99], [Yi03]: 1) Diversification: Diversity with respect to the focus of the activities performed by the interviewees was a key criterion when selecting the participants of the study. 2) Methodological triangulation: We used different methods to analyse the data (quantitative and qualitative techniques, as described in Section 3.3). 3) Explanatory triangulation: by trying out several explanations for all results in Section 4. For example, the result, that the requirements specification document is a key information source for testers can be confirmed by several facts. First, asked for main problems in the testing process, almost all interviewees indicate the poor quality of the requirements specification. In addition, asked for required input for different decisions, the interviewees indicate the requirements specification as an important input for almost all decisions. Finally, asked for a "wish list", the testers indicate that the quality of the requirements specification is more important than other sources of information. Based on these three facts, we conclude that the requirements specification is an important information source for

testers. Nevertheless, organisations with a higher degree of test automation or which use more formal models (e.g. embedded area) may show different results.

7. Conclusions and Future Work

In this research work, we presented the results of an exploratory study performed during the SIKOSA project with experienced testers with the aim to analyze the information flow during testing as the starting point for test process improvements. This work served as basis for the definition of the PAT3-Approach [IP06] as part of the whole SIKOSA methodology. The PAT3 Approach captures testing **experience and knowledge** in form of patterns. PAT3 defines five pattern categories (process patterns, automation patterns, transformation patterns, testability patterns, traceability patterns) which improve the interface between requirements engineering and testing.

The main results of our study regarding the research questions formulated in Section 3.1 can be summarized as follows:

The requirements specification is, not surprisingly, the document used most frequently during testing (Question 1). This document is used as input for all decisions to be made. However, there is another information source which is almost equally valuable: previously found defects. In addition, the requirements engineer and the project manager are roles consulted most frequently by testers (Question 2). Surprisingly, testers mention a high communication overhead during test execution. This fact is an indicator for poor quality of the requirements specification, confirmed as a major problem during testing by almost all interviewees. Experience plays an important role, and the definition of test data requires by far the most experience (Question 3). Moreover, decisions related to TP&C and TCE require much experience. At first glance, the latter is unexpected, but since most organizations do not define a test strategy, evaluation is not easy in the absence of operational goals. As expected, test execution requires little experience and is consequently well suited to be automated.

This exploratory study gives first indications for hypotheses we aim to verify in subsequent empirical studies. Based on our results, we can formulate the following hypotheses. **H1:** Previously found defects are good indicators and predictors for future defects. **H2:** Embedding the user into the testing process (particularly into the prioritization and reviewing the test cases) increases the efficiency of the testing process. **H3:** Testing decisions and activities require more system specific than general experience. **H4:** Combined approaches integrating experience into traditional test design techniques lead to better test cases.

Acknowledgements

We would like to thank all the interviewees for their cooperation and help by providing information and insight into documents.

8. References

[BIP07a] Borner, L.; Illes-Seifert, T.; Paech, B.: Entscheidungen im Testprozess, Software Engineering Konferenz (SE 2007), Hamburg, March 27 - 30, 2OO7.

[BIP07b] Borner, L.; Illes-Seifert, T.; Paech, B.: The Testing Process - A Decision Based Approach, In: in: Proceedings of the The Second International Conference on Software Engineering Advances (ICSEA 2007), IEEE Computer Society, Cap Esterel, France, 25.-31. August, 2007. Cap Esterel, French Riviera, France, August 25 - 31, 2OO7

[BSC96] Burnstein, I.; Suwannasart, T.; C. R Carlson: Developing a Testing Maturity Model for Software Test Process Evaluation and Improvement, Proceedings of the IEEE International Test Conference on Test and Design Validity, 1996.

[Da05] Dahlstedt, A.: Guidelines Regarding Requirements Engineering Practices in order to Facilitate System Testing, the Proceeding of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality, Porto, Portugal, 13-14 June 2005.

[Do97] Dowson, M.: The Ariane 5 software failure. SIGSOFT Softw. Eng. Notes, ACM Press, Mar. 1997, 22, 2, pp. 84.

[IEEE98] IEEE Std. 829-1998, IEEE standard for software test documentation, Software Engineering Technical Committee of the IEEE Computer Society, USA, 1998.

[IHPR05] Illes, T.; Herrmann, A.; Paech, B; Rückert, J.: Criteria for Software Testing Tool Evaluation. A Task Oriented View. Proceedings of the 3rd World Congress for Software Quality, 2005

[IP08] Illes, T.; Paech, B.: On the Role of Communication, Documentation and Experience during Testing - Results of an Interview Study, SWEHD-TR-2008-02, http://www-swe.informatik.uni-heidelberg.de/research/publications/reports.htm, 2008.

[IP06] Illes, T.; Paech, B.: From "V" to "U" or: How Can We Bridge the V-Gap Between Requirements and Test?, Software & Systems Quality Conferences 2006, on May 10th 2006 in Düsseldorf.

[ISTQB05] International Software Testing Qualifications Board, ISTQB Standard Glossary of Terms used in Software Testing V1.1, September 2005.

[KP02] Koomen, T.; Pol, M.: Test Process Improvement: A step-by-step guide to structured testing. Addison-Wesley, 1999.

[KP03] Kitchenham, B; Pfleeger, S. L.: Principles of survey research part 6: data analysis, SIGSOFT Softw. Eng. Notes 28, 2, Mar. 2003, pp 24-27.

[MP02] Mosley, D. J.; Posey, D. J.: Just Enough Software Test Automation, Prentice Hall, July 2002.

[Se99] Seaman, C.B.: Qualitative Methods in Empirical Studies of Software Engineering, IEEE Transactions on Software Engineering, 25(4), July/August 1999, pp. 557-572.

[Se02] Seaman, C.B.: The Information Gathering Strategies of Software Maintainers, In Proceedings of the international Conference on Software Maintenance ICSM. IEEE Computer Society, Washington, DC.

[SLS06] Spillner, A.; Linz, T.; Schaefer, H.: Software Testing Foundations - A Study Guide for the Certified Tester Exam - Foundation Level - ISTQB compliant, Dpunkt Verlag, 2006.

[TL01] Tjortjis, C.; Layzell, P.: Expert Maintainers' Strategies and Needs when Understanding Software: A Case Study Approach. In Proceedings of the Eighth Asia-Pacific on Software Engineering Conference (APSEC). IEEE Computer Society, Washington, DC, 2001.

[Yi03] R.K. Yin, “Case Study Research, Design and Methods”, SAGE Publications, USA, 2003.

New Applications for Wikis in Software Engineering

Michael Geisser[1], Hans-Jörg Happel[2], Tobias Hildenbrand[1],
Axel Korthaus[3], and Stefan Seedorf[3]

[1]: University of Mannheim, Department of Information Systems I
{geisser|hildenbrand}@wi.uni-mannheim.de
[2]: Research Center for Information Technologies, Information Process Engineering
happel@fzi.de
[3]: University of Mannheim, Chair in Information Systems III
{korthaus|seedorf}@uni-mannheim.de

Abstract: Within software development, wikis are currently mainly used for brainstorming and documentation purposes or error management and project coordination. This article describes four advanced application scenarios for wiki support in software development processes: Requirements Engineering, Traceability and Rationale Management, Architectural Knowledge Sharing, and Lessons Learned Management in a distributed knowledge infrastructure. Finally, we will give a conclusion by summarizing the main advantages and drawbacks of the presented innovative uses of wikis in software engineering.

1 Introduction

Wikis are easy to use but powerful tools for collaborative knowledge formation and knowledge sharing. Due to their widespread adoption in Web communities, wikis are being increasingly employed in enterprise settings [MaWY06], especially in software development projects. Their current uses in software development range from brainstorming and documentation to error management and project coordination. However, there is still potential for a more extensive support of software processes, which has not been fully tapped yet. The question is therefore how other core and supporting activities in Software Engineering (SE) can benefit from innovative uses of wikis.

This article describes new applications of wikis to support various SE activities. First, we give an outline of wikis in general and the more recent "semantic wikis", which enhance traditional wikis to allow machine-interpretable content marking and contribute to an integrated process support. After a short overview of wiki applications in SE, we describe four concrete application scenarios and corresponding prototype implementations: Requirements Engineering (RE), Traceability and Rationale Management (TRM), Architectural Knowledge Sharing, and Lessons Learned Management in a distributed knowledge infrastructure.

The process of RE, the initial phase of a software development project, comprises requirements elicitation, analysis, specification, and validation for the planned system. For this purpose, wikis are a lightweight and agile alternative to many commercial and par-

tially very complex solutions. In RE and the following phases of software design, interdependencies of requirements and between requirements and the resulting artifacts, as well as the rationale they are based on, should be made accessible (TRM). For the collection and cross-linking of such information, a systematic TRM-process is necessary and Wiki-systems can be appropriate implementation tools due to their usability and hypertext capabilities. They can also serve for architectural knowledge sharing by enabling multi-perspective, collaborative documentation and exchange of different views of software architectures. By leveraging the semantic power of semantic wikis, additional value can be achieved, e.g. with regard to advanced browsing, querying, and searching of the knowledge base. The Ontobrowse semantic wiki, which is described in section 3.4, is an example of a semantic wiki tool supporting this application scenario. Finally, we present ToMaWiki, a semantic wiki that can be used as a front end to a distributed, topic map-based knowledge infrastructure and can be employed, for example, to implement a Lessons Learned Management System (LLMS) for storing, synthesizing, and reusing software project knowledge according to the Experience Factory organizational concept [BaCR94]. We conclude by summarizing the main advantages and drawbacks of the presented innovative uses of wikis in software engineering.

The methods and tools introduced in this paper have mainly been developed in the context of the CollaBaWue research project[1]. The first two examples are based on the wiki system of CodeBeamer[2], a collaborative software development platform from Intland, but can be transferred to most other wiki systems. The Ontobrowse semantic wiki has been developed from scratch, and ToMaWiki uses the open source semantic wiki Ikewiki[3] as its foundation.

2 Wikis

Wiki (Hawaiian for “fast”) is a term used to denote software programs that provide an easy method for multi-user web-based cooperation. It is important to distinguish between the software required to operate a wiki (“wiki engine”) and a wiki instance (“wiki”, “wiki installation”). The well-known online encyclopaedia Wikipedia, for example, uses the MediaWiki software as its wiki-engine. Although most wiki engines are available as open source, the number of commercial products and hosting/ASP vendors in the market (e.g. SocialText, JotSpot) has been rising lately.

[1] http://www.collabawue.de
[2] http://www.codebeamer.com
[3] http://sourceforge.net/projects/ikewiki/

A wiki is basically made up of several distinct pages, which are interconnected via hyperlinks – comparable to a "small version" of the World Wide Web, which is also made up of linked documents. As in web content management systems (WCMS), wiki pages can be directly created and modified using a Web browser. However, they do not have fixed content categories or limited user rights. Generally, all users can edit the contents of a wiki by using a simple wiki language or a WYSIWYG editor. When a user edits a page, a revision history is saved so that previous versions can always be restored. Thus, social control replaces complicated rights management in WCMS [EbGl05].

Since the basic functions of wikis are as simple as those of common web-based email services, they allow an easy start. Although this leads to fast results and user satisfaction in editing text-based knowledge, it provides only little help in structuring this knowledge. Thus, with increasing use of the wikis, they tend towards complexity and content sprawling [MaWY06]. This leads us to the main drawback of wikis: Although the content of a wiki page might provide structure and meaning to a human reader, it does not possess any machine-interpretable semantics. Advanced knowledge management features such as semantic search and metadata-based filtering are thus not available in traditional wikis. However, a site like Wikipedia could heavily benefit from structuring content with additional metadata, which can be used to derive a knowledge model [VöKr06]. In that way, explicit but informal knowledge embedded in a page could be transferred into machine-processible knowledge, which can then be used for semantic queries.

This extension towards so-called "semantic wikis" has been realized in several projects, either by implementing a completely new wiki engine or by extending an existing one. Although the core idea of all semantic wikis is to provide a machine-processible knowledge model described in the wiki pages, they vastly differ in terms of required user experience and knowledge representation languages. For example, the Semantic MediaWiki adds some extra syntax for the semantic annotations to the wiki markup language [VöKr06]. It therefore realizes a very open approach where a user can optionally add semantic markup. Our Ontobrowse semantic wiki, on the other hand, interprets every page as an entity, which may be a concept, object property, datatype property, or individual object [HaSe07]. Thus, its semantics are defined in a more rigorous style, which enables us to acquire semantic data from external knowledge bases. In sections 3.4 and 3.5, two applications are described and the added value of semantic wikis is demonstrated.

3 Selected Applications of Wikis in Software Engineering

The wiki technology was invented and originally used by software developers. The first wiki, initiated by Ward Cunningham[4] in 1995, has served as a knowledge repository for design patterns. Since then, software development has been remaining one of the most important application areas of the wiki technology [Lour06]. Frequently, a small group of users sets up a wiki, which is subsequently used by an ever-growing number of em-

[4] http://www.c2.com/

ployees. The simple availability and installation of wiki engines is especially helpful in this respect.

Aside from open source development communities such as the Apache Foundation and smaller enterprises, large software enterprises such as SAP, Novell or Yahoo have adopted wikis as well (cp. [HGN06]). General applications in SE comprise knowledge transfer, technical documentation, quality and process management, release planning, and error tracing [BaMe05, MaWY06, TWIK06]. However, for specific, well-structured content, traditional wikis often reach their limits with their core functionality. Thus, add-ons with specific functionalities are already available for problems such as source code documentation [AgDa05, JoSW05] or error tracing [Edge06].

Subsequently, we describe two advanced application domains of standard wikis in software development processes: Requirements Engineering as well as Traceability and Rationale Management.

3.1 Requirements Engineering

RE, the systematic elicitation, analysis, specification, validation, and management of software requirements, is usually being performed either with heavyweight RE tools or even with common office software. Traditional RE tools like RequisitePro and DOORS as well as office products are widely spread, but were originally not designed for an internet-based environment. This causes, for instance, performance problems and inefficient processes (cp. [IHGH07]). For larger software projects, distributed RE via office software can only be regarded as a makeshift solution. Documents often are only distributed via email but do not get assigned centralized version numbers.

There are no integrated methods for distributed RE so far. Therefore, we present DisIRE (Distributed Internet Based Requirements Engineering), a wiki-supported method for distributed, internet-based RE with specific tool support, which can be integrated in existing development platforms. DisIRE combines successful RE approaches and extends them to create a solid theoretical and empirically valid methodical framework[5]. Moreover, DisIRE offers the first integrated RE method for a distributed environment where requirements are explicitly taken into account. This method allows a system vendor to perform a largely distributed and at the same time systematically RE process. A moderator accompanies all parties through the process as described below.

3.1.1 Requirement Elicitation and Analysis

DisIRE follows immediately after the feasibility study. At this point, aside from a vision of the planned system, there is also the certainty that the project has realistic goals. Usually the first step includes the requirements elicitation and analysis, which is described in this section.

[5] Such approaches are: EasyWinWin [Grün03], QuantitativeWinWin [RuEP03] and the Cost-Value-Approach [KaRy97]. A detailed description of the DisIRE method can be found in [GHHR07].

As face-to-face meetings of all parties involved lead to better results in distributed work contexts as well (compare [GeHi06]), an initial meeting takes place with as many participants as possible, but no less than one representative from each location and organizational unit involved. The goal is to make sure that all attendants get a uniform picture of the planned system.

After the initial meeting, the requirements are elicited asynchronously and then analyzed. An overview of the different roles with their corresponding tasks is shown in Table 1. When all requirements are commented, consolidated and verified, and it seems like no further comments or requirements will be communicated, the moderator can freeze these requirements.

Roles	Task
Representative of the customers	• Transmission of requirements • Commenting on requirements • Identifying unclear technical terminology • Definition of unclear technical terminology • Extending and specifying vision
Moderator	• Force specification • Consolidation and categorization of requirements • Identification of unclear technical terminology • Extending and specifying vision
Software Engineer	• Commenting on requirements • Feasibility testing • Identification of unclear technical terminology

Table 1: Roles with their corresponding tasks in requirement elicitation and analysis with DisIRE

A wiki is especially suitable for the process described above: for each initial requirement a distinct wiki page is created where related comments follow that are consolidated directly in the description of the requirement. An integrated versioning system guarantees that all changes can be traced on each page. Furthermore, articles or particular domains are locked during editing to avoid conflicts. Another advantage of wiki systems is the possibility to use an integrative glossary: Technical terminology, which is used for the description of the requirements and which needs further explanation, can be marked through wiki-links and be defined in a glossary. All other asynchronous activities can also be thoroughly supported by wiki technology. CodeBeamer, a collaborative software development platform from Intland Software[6] [RBGH07], has an integrated wiki module and was used within the CollaBaWue project for an integral support of the DisIRE process [GHHR07].

[6] http://www.intland.com/

3.1.2 Requirements Selection

Requirements elicitation and analysis produces a consolidated list of requirements, whose full implementation, however, is not always economically rational. The list may contain requirements that cause an elaborate implementation but bring little gain. Hence, those requirements should be selected whose implementation appears to be rational under economic criteria. Since no wiki technology is used in this step within DisIRE, this activity will not be presented here (see [GHHR07]).

3.1.3 Requirements Specification and Validation

Subsequent to requirements selection, in order to achieve a certain degree of specification for the later steps, the selected requirements have to be specified. If, as in DisIRE, stakeholders on the customers' side are asked directly about their requirements, generally functional requirements are concerned. Use cases are particularly suitable for the specification of this type of requirements. Hence, for an internet-based specification, templates for such use cases are made available directly in CodeBeamer's requirements tracker. As a result, software engineers can use these templates on every workstation connected to the Internet to specify the requirements. Finally, the specified requirements have to be validated by the stakeholders on the customers' side to make sure that during the specification no information has been lost and no misunderstandings have occurred. This is done with the help of an integrated comments function of CodeBeamer's Requirements Tracker.

To enable versioning of specifications and respective comments as well as parallel editing, a wiki was directly integrated into CodeBeamer's requirements tracker and its comments function. In this way, wiki content and other tracker items and artifacts from CodeBeamer projects can be referenced very easily and quickly.

3.2 Traceability and Rationale Management

In the next phases of software design, interdependencies of requirements and between requirements and the resulting artifacts, as well as the rationale they are based on, should be made available. This entails, above all, the administration of different interdependencies and the exact reasons and alternatives for each decision, such as modifications, in every step of the process [DMMP06]. The more distributed stakeholders are, the more complicated this task becomes [RaJa01]. Thus, based on the requirements specifications in RE, "Wiki-TRaM" will now be introduced, a Wiki-based TRM-process for information acquisition, management, visualization, and analysis. Wiki-TRaM is made up of two disciplines, "Collection and Management" and "Visualization and Analysis", at which we will take a closer look below.

3.2.1 Collection and Management of Traceability and Rationale Information

Although DisIRE allows systematic requirements elicitation, analysis, and selection, the specification is in most cases not yet complete and stable. Therefore, an equally systematic change management is also necessary. That is only possible if traceability of the development process of the requirements (source traceability), of the dependencies of requirements among each other (requirements traceability) and of the resulting artifacts (design traceability) is given [Somm04]. The dependencies of pre-specification artifacts are vitally important for the requirements and change management within the framework of RE. These consist of the requirements descriptions and comments, which are linked with the specified requirements from an issue tracker through a corresponding wiki page. Issue trackers are used for managing task descriptions and status. These in turn are deposited together with the dependencies among the specified requirements, the cost and value ratios, as well as the reasons for the selection of the decisions in CodeBeamer's issue tracker. Thus, apart from the traceability and rationale information, the implementation status of the individual requirements can also be managed and the respective artifacts versioned.

In the next step of the software project, the post-specification phase, relationships between the specified requirements and the resulting artifacts, e.g. design models, source code, and test cases, as well as all essential decisions and their rationale are recorded. For this, wikis offer the necessary functionality for content linking and commenting.

This allows for easier customization, maintenance, and reuse of the system and faster training of new project staff. To this end, different trackers for each task field (RE, architecture design, development, etc.) are used on the CodeBeamer platform and the respective tasks are linked through their association mechanism or through wiki links. Standardized relationships between tracker items themselves, artifacts, and external URLs are recorded using associations. Besides CodeBeamer's own document management system, artifacts, e.g. source code, from external repositories like Subversion can also be referenced. All changes to an artifact require the posting of a comment, or in the case of SVN a commented "commit". Due to the use of wiki comments, relationships to all CodeBeamer artifacts and external resources can be established. With the help of these mechanisms (associations and wiki links), traceability and rationale information can be documented and linked during the entire project.

3.2.2 Visualization and Analysis of Traceability and Rationale Information

Based on detailed recording of the information above, a heterogeneous "trace network" emerges, which contains relationships between process steps (i.e. tracker items), artifacts, and persons involved. For a clearer visualization and a more effective analysis of this traceability information, the tool "TraVis" was developed (TraVis = Trace Visualization). It is a Java-application, which extracts the respective basic information using CodeBeamer's interface and displays it according to role-based filters. These filters allow different views for developers, consultants, project managers, etc. The displayed artifact categories and relationships can vary. Additionally, individual artifacts, tracker

items, or users with their direct and indirect traceability network can be extracted. Further details of this tool will be left out here, since it only processes data from a wiki.

In comparison to traditional SE tools, wikis allow for a more collaborative and agile approach to certain tasks such as requirements elicitation and trace capturing. However, in order to achieve increased overall SE productivity, they need to be fully integrated in to existing development environments such as *source code* versioning, issue tracking, and document management [Hild08].

4 Selected Applications of Semantic Wikis in Software Engineering

As described in section 2, semantic wikis provide new ways of structuring and processing knowledge. Here, we introduce two semantic wikis and their respective applications in software engineering: First, Ontobrowse semantic wiki enables the sharing of architectural knowledge. Second, ToMaWiki supports Lessons Learned Management in a distributed knowledge infrastructure.

4.1 Architectural Knowledge Sharing

The bridge between requirements engineering and concrete design is software architecture. Many stakeholders are involved in the development and maintenance of architecture. Hence, integrated tool support is difficult, because various knowledge needs have to be catered for. On the one hand, developers want technical support and guidance for their implementation task at hand. On the other hand, architects and business analysts demand tools for analysis and documentation. Thus, the actual representation of an architecture instance is usually split up into several “views”, such as functional, physical or logical [Kruc95]. Most of these views can be assigned a certain purpose, such as quality, communication, analysis, or reuse [BaCK03, Bosc00]. Software architecture documentation should include all these views. However, existing tools are often not flexible enough to support the requirements of both groups appropriately, which leads to scattering of architectural knowledge into different information spaces.

What is sought after is a solution that offers both flexibility in documentation and collaboration and a formal basis for leveraging machine-interpretable semantics. Although this requirement sounds contradictory at first glance, semantic wikis are a promising candidate for solving this trade-off. From our point of view, semantic wikis are well-suited to bridge the gap between technical and business documentation. First, they encourage collaborative documentation and information exchange. Second, they provide the means for processing machine-interpretable knowledge, which is required for handling technical descriptions. Within the CollaBaWue project, the Ontobrowse semantic wiki has been specifically developed for the sharing of architectural knowledge. It provides the following main features [HaSe07]:

- Defining a knowledge structure using ontologies
- Browsing, querying and searching of a knowledge base

- Combining informal and formal documentation
- Integrating asserted knowledge from external sources
- Consistency checking with rules

The semantic wiki has been implemented employing the Web Ontology Language (OWL) as its knowledge representation format. The knowledge base can be configured to use either Jena[7] or KAON2[8] as reasoner. There is also experimental support for enforcing architectural rules using the Semantic Web Rule Language (SWRL).

An application scenario is the documentation of a service-oriented architecture (SOA). SOA is an architectural style, which propagates the orchestration of business processes from independent, loosely-coupled services. Service-orientation leads to a rising level of alignment between business processes and IT implementation. Therefore, it becomes more important to monitor and guide the development of the services landscape [HaSe07]. Because the black-box specification of a service encourages a higher decoupling of software systems, responsibilities are shared by different service providers – together with the associated business and technical knowledge.

In order to integrate architectural descriptions of a SOA into the wiki, one has to perform two distinct steps: First, the users of the wiki need to agree upon a unifying *SOA ontology*. The ontology defines the terminology of the architecture together with relations and constraints. The terminology can usually be displayed as a concept hierarchy, with a generic concept such as "SOAElement" subsuming more specific concepts such as "Service" and "BusinessObject" (see Figure 1).

Figure 1: A generic concept in Ontobrowse

[7] http://jena.sourceforge.net/
[8] http://kaon2.semanticweb.org

The ontology reduces conceptual ambiguity and enables information integration. Second, a plug-in has to be defined, which performs a mapping of architectural knowledge in a given format to the SOA ontology. The plugin can then be configured to crawl for matching specification files in one or more directories. As a result, a service specification mapped from a WSDL file is displayed on a page together with a property description (see Figure 2). Once an entity has been imported into the knowledge base, it can be augmented with textual descriptions and additional metadata. The same pattern can be repeated for other types of knowledge, such as requirements or business process specifications.

Ontobrowse addresses key issues in the documentation and maintenance of software architectures. Previously, the different views of an architectural instance were maintained in separate information spaces. The semantic wiki enables the integration of both business-oriented and technical knowledge, thus serving as a single point of information for all stakeholders. Due to the underlying formal representation based on ontologies, searching and querying can be significantly improved. At the same time, the plugin infrastructure makes it possible to integrate knowledge from external sources. Architectural descriptions such as service specifications in two different formats can be mapped to the same ontology. Finally, the conceptual structure is modular so that additional ontologies can be added at any time, e.g. to describe the organizational structure. These extension features are particularly important in distributed development settings (cf. [Cock96]), where the participants have to share their knowledge with other developers.

Figure 2: An individual object in Ontobrowse

4.2 Software Engineering Lessons Learned Management

Enterprise software development today is most often organized in a distributed way, involving different sites and organizations in a potentially globally distributed "software eco system". Generally, it will be impracticable to standardize a comprehensive, overall ontology on which knowledge management is based for all players and collaboration partners in the software eco system. On the contrary, local ontologies will have to be used in loosely coupled systems, which, however, must be matched in order to reuse knowledge across single sites. To support this scenario, we have implemented the backbone infrastructure component for a distributed software development knowledge management system, which we call a "Topic Grid". It provides the functionality required to be able to exchange knowledge structures between various applications.

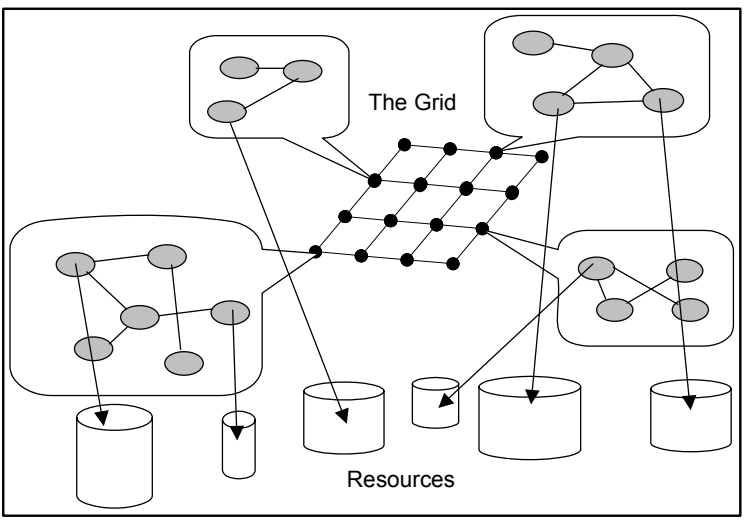

Figure 3: Sketch of the Topic Grid Interconnecting Different Semantic Networks of Resources

In [KoHi03], the basic idea of the Topic Grid is described as being a network of nodes each of which provides its own knowledge base in the form of one or more Topic Maps (see Figure 3). It can be seen as a superimposed semantic network over a multitude of heterogeneous electronic documents and information resources (maintained inside and outside the company) providing relevant information to software developers. The Topic Maps are made available for queries from other nodes by means of a standardized protocol. In the Topic Grid, a client is capable of querying all knowledge bases belonging to a certain group in parallel, so that a single, transparent view on the knowledge bases and knowledge structures is created. The Topic Grid aims at providing applications with a homogeneous view of distributed Topic Maps pretending that the user works with one big, connected Topic Map. This is evocative of a typical notion of grid computing, namely the virtualization of a multitude of physically separated computer systems. In

[KAHS06] and [KoAH08], we describe the design and implementation of a Java-based Topic Grid prototype using an access protocol stack for distributed Topic Maps in a network to realize the idea of the Topic Grid.

To use the Topic Grid infrastructure in a real-world context, we have implemented a knowledge management system based on this infrastructure, which will be used to support software developers collaboratively working together at different locations to perform distributed software engineering activities. As Topic Map-aware applications to be integrated via the Topic Grid in that setting, we envision, among others, semantic wikis. A first prototype of a semantic wiki ("ToMaWiki") as front end to the Topic Grid has already been built. ToMaWiki not only provides typical semantic wiki functionality but also a graph-based navigation feature for displaying fragments of the complete Topic Grid (cf. Figure 4; [KoSc06]). With the help of this feature, users can quickly jump to relevant knowledge topics whose details are then displayed as regular wiki pages. Software developers can use this kind of Topic Grid-aware semantic wikis in an ad hoc way to document software engineering knowledge or project experience on the fly, which can subsequently be accessed from other wikis or applications on the Topic Grid. Software developers can use the wikis, for example, to exchange ideas on domain and technical issues, to store decisions made and their rationales, to share social information, to identify and locate experts, to self-coordinate collaborative work tasks, and to track progresses on project tasks (cf. [ChMa05]).

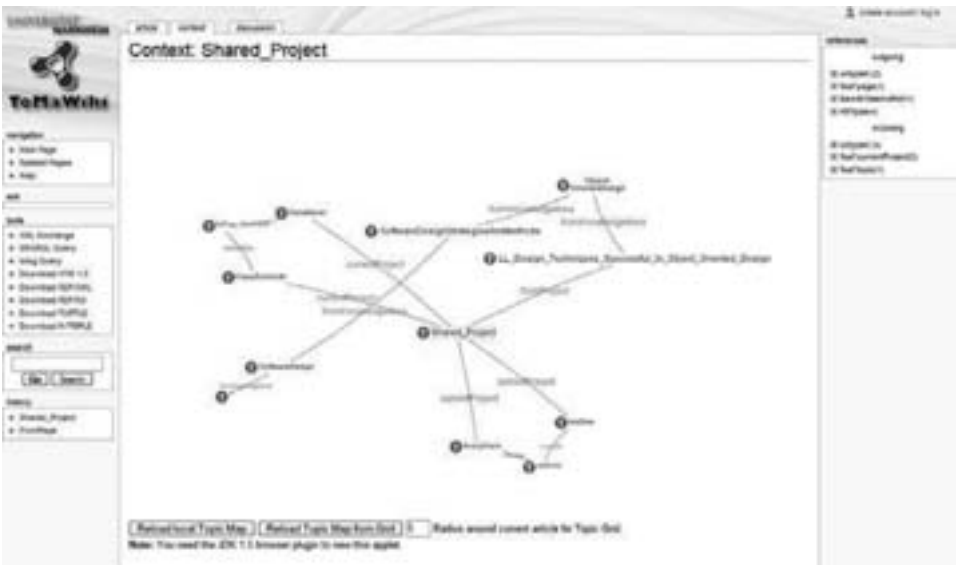

Figure 4: Screenshot of the Graphical Navigation Feature of the Prototypical Semantic Wiki Front End to the Topic Grid

By combining the Topic Grid and the ToMaWiki semantic wiki using Topic Map technology in the background to provide an ontology that helps to annotate wiki pages and links with machine-interpretable metadata, we were able to achieve some important goals. First, by sticking to the well-known wiki technology, we provide a very lightweight approach to knowledge provisioning that does not significantly hinder the core software development process. By using wikis with semantic enhancements, we achieve the benefits leveraged by ontology-based semantic technologies, like improved semantic searches. Based on the Topic Grid infrastructure in the background and the Topic Map technology's concept of merging and identity, which allows automated integration of topic maps from diverse sources into a coherent new topic map, we enable distributed teams at different sites to use at least partially different ontologies while remaining able to search the complete Topic Map information space.

As a first concrete application scenario, we think of developers documenting their current development tasks and "lessons learned" using the semantic wiki to implement a kind of "self-organized experience factory unit" (cf. [ChMa05]). A suitable ontology for annotating their information serves as an entry point into the Topic Grid. New pages edited in the wiki can then automatically be added in a semantically structured way to the local Topic Map managed at the developer's site and thus become part of the body of information in the Topic Grid. This body of information, however, will also be increased by many other Topic Map-aware applications. We have already implemented several prototypical applications for this purpose, e.g. a tool to generate a Topic Map semi-automatically from a conventional document index provided by the Lucene tool [KoKS04] or a Topic Map-based web application modelling knowledge about refactoring tasks.

The semantic search functionality offered by the semantic wiki is not limited to local wiki contents, but accesses the whole Topic Grid, hence tapping a very comprehensive knowledge base in order to satisfy the developer's information needs that had arisen for him to successfully perform his current development activity. For example, questions like "What issues are involved when combining EJBs and JDBC?" can lead to the system providing links to information resources containing the relevant information or even contact information of experienced colleagues, lesson learned stories, relevant design patterns, newsgroup postings etc.

5 Conclusion and Outlook

New Web-based collaboration technologies, such as wikis, which let users easily publish and share content, are thriving recently not only in the context of Web 2.0 and the Social Web, but also in various business contexts, and especially in the domain of software development. Although they lack sophistication, the strength of traditional wikis as a platform for collaborative authoring and sharing of contents lies in their simplicity and efficiency. As has been demonstrated in this paper, the "social software" wiki (cf. [Bäch06]) can be of great use for software engineering, not only in traditional application areas such as distributed documentation. With the continuing evolution of wiki technology and the development of new methodological approaches, other usage scenarios in software engineering can be opened up. We have presented Requirements Engineering as well as Traceability and Rationale Management as two concrete examples of software development activities that can benefit from the methodical employment of wiki technology.

Both application cases and their wiki-based implementations make it clear that wikis are in no way limited to open-source development, but are also an interesting and flexible approach for the support of enterprise processes. In the open-source domain, wikis have so far been employed as an "agile" tool for knowledge management and asynchronous collaboration. Aside from RE and TRM, other application areas of software development should be entered in the future, since—with the increasing specialization of roles in the development process—an ever growing amount of information requires distributed storing and automatic processing. Moreover, an integration of the different wiki application areas in SE, e.g. on the basis of a collaboration platform, needs to be achieved in order to provide full productivity caused by different wiki-based solutions. Integrated platforms such as CodeBeamer (cp. section 3) provide one central wiki engine for different SE tasks an need to be further enhanced with respect to semantic annotation and reasoning.

To this end, however, an alleviation of the weak points of traditional wikis regarding machine-interpretable structuring of contained knowledge is necessary. In traditional wikis, a metadata infrastructure is absent, and information is usually handled in an ad hoc fashion. Semantic wikis represent an innovation that aims at expressing content in machine-processible forms that enhance search precision and logical reasoning. Based on two examples of prototypical semantic wiki implementations for architectural knowledge sharing and software lessons learned management, we have demonstrated the potential advantageousness of semantic wiki technology for distributed software development scenarios. Although the added value that can be gained from semantic approaches in software development is not generally contested, their long-term success will depend considerably on the question how seamlessly they can be integrated in the core business processes without being perceived as causing too much undesired work overhead.

References

[AgDa05] Aguiar, A.; David, G.: WikiWiki weaving heterogeneous software artifacts. In: Proc. of the 2005 International Symposium on Wikis, San Diego, CA, 2005, pp. 67-74.

[BaMe05] Bachmann, F.; Merson, P.: Experience Using the Web-Based Tool Wiki for Architecture Documentation. Technical Note CMU/SEI-2005-TN-041. September 2005.

[Bäch06] Bächle, M.: Social Software. In: Informatik Spektrum, 2006, 29, pp. 121-124.

[BaCR94] Basili, V.R.; Caldiera, G.; Rombach, D.: Experience Factory. In: Marciniak, J.J. (ed.), Encyclopedia of Software Engineering, vol. 1, 1994, John Wiley & Sons, pp. 469-476.

[BaCK03] Bass, L.; Clements, P.; Kazman, R.: Software Architecture in Practice. 2. Addison Wesley, 2003.

[BeHL01] Berners-Lee, T.; Hendler J.; Lassila, O.: The Semantic Web. In: Scientific American, 284, 5, 2001.

[Bosc00] Bosch, J.: Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley Publishing Co., 2000.

[ChMa05] Chau, T. and Maurer, F.: A Case Study of Wiki-based Experience Repository at a Medium-sized Software Company. In: Proc. of 3rd Int. Conf. on Knowledge Capture (K-CAP '05), Oct. 2-5, 2005, Banff, Alberta, Canada, pp. 185-186.

[Cock96] Cockburn, A.: The interaction of social issues and software architecture. In: Commun. ACM 39, October, Nr. 10, 1996, pp. 40-46.

[DeRe05] Decker, B.; Rech, J.; Ras, E.; Klein, B.; Hoecht, C.: Self-organized Reuse of Software Engineering Knowledge supported by Semantic Wikis. In: Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE). November 2005.

[DMMP06] Dutoit, A.H.; McCall, R.; Mistrik, I.; Paech, B. (Hrsg.) Rationale Management in Software Engineering, Springer Verlag, 2006.

[EbGl05] Ebersbach, A.; Glaser, M.: Wiki. In: Informatik Spektrum, (28), 2005, pp. 131-135.

[Edge06] Edgewell.org: The Trac User and Administration Guide, URL: http://trac.edgewall.org/wiki/TracGuide (26.09.2006), 2006.

[GeHi06] Geisser, M.; Hildenbrand, T.: A Method for Collaborative Requirements Elicitation and Decision-Supported Requirements Analysis. In: Ochoa, S.F. und Roman, G.-C. (eds): IFIP Int. Federation for Information Processing, Advanced Software Engineering: Expanding the Frontiers of Software Technology, 2006, pp. 108-122.

[GHHR07] Geisser, M.; Heinzl, A.; Hildenbrand, T.; Rothlauf, F.: Verteiltes, internetbasiertes Requirements-Engineering. In: WIRTSCHAFTSINFORMATIK, Volume 49 (3), 2007, pp 199-207.

[Grün03] Grünbacher, P.: EasyWinWin: Eine groupware-unterstützte Methode zur Erhebung und Verhandlung von Anforderungen. In: Softwaretechnik-Trends der Gesellschaft für Informatik, 23, 2003.

[HaSe07] Happel, H.-J. and Seedorf, S.: Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures. In: Proc. of the 19th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Boston, USA, July 9-11, 2007, pp. 506-512.

[HGN06] Hildenbrand, T.; Geisser, M.; Nospers, M.: Die Übertragbarkeit der Open Source-Entwicklungsmethodik in die Unternehmenspraxis". In: Softwaretechnik-Trends, Volume 26 (1), 2006, pp. 37-42.

[Hild08] Hildenbrand, T.: Improving Traceability in Distributed Collaborative Software Development - A Design Science Approach. Dissertation Thesis, University of Mannheim, Germany, 2008.

[IHGH07] Illes-Seifert, T.; Herrmann, A.; Geisser, M.; Hildenbrand, T.: The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey". In: Proc. of the 1st Int. Global Requirements Engineering Workshop (GREW´07), 2007, pp. 55-65, Munich, Germany.

[JoSW05] John, M.; Jugel, M.; Schmidt, S.; Wloka, J.: Wikis in der Softwareentwicklung helfen. In: Java Magazin, 7, 2005, pp. 88-91.

[KaRy97] Karlsson, J.; Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. In: IEEE Software, 14, 5, 1997, pp. 67-74.

[KoAH08] Korthaus, A., Aleksy, M., and Henke, S.: A Distributed Knowledge Management Infrastructure Based on a Topic Map Grid. In: Int. Journal of High Performance Computing and Networking (IJHPCN), Interscience Publishers, vol. 6, issue 2/3, 2008.

[KAHS06] Korthaus, A., Aleksy, M., Henke, S., and Schader, M.: A Distributed Topic Map Architecture for Enterprise Knowledge Management. In: Proc. of the 1st IEEE/ACIS Workshop on Component-Based Software Engineering, Software Architecture and Reuse (COMSAR ’06), July 10-12, Honolulu, Hawaii, USA, 2006.

[KoHi03] Korthaus, A., Hildenbrand, T.,: Creating a Java- and CORBA-Based Enterprise Knowledge Grid Using Topic Maps. In: Cheung, W.K., and Ye, Y. (eds.), Proc. Workshop on Knowledge Grid and Grid Intelligence”, Halifax, Canada, 2003, pp. 207-218.

[KoKS04] Korthaus, A., Köhler, C. and Schader, M.: (Semi-) Automatic Topic Map Generation from a Conventional Document Index. In: Proc. IASTED Int. Conf. on Knowledge Sharing and Collaborative Engineering (KSCE 2004), St. Thomas, US Virgin Islands, Nov. 22-24, 2004, pp. 101-108.

[KoSc06] Korthaus, A. and Schader, M.: Using a Topic Grid and Semantic Wikis for Ontology-Based Distributed Knowledge Management in Enterprise Software Development Processes. In: Proc. of the IEEE Int. Vocabularies, Ontologies and Rules for the Enterprise Workshop (VORTE 2006) / Tenth IEEE Int. EDOC Conference, 16. October, Hong Kong, China, 2006.

[Kruc95] Kruchten, P.: The 4+1 View Model of Architecture. In: IEEE Softw. 12 November, Nr. 6, 1995, pp. 42-50.

[Lour06] Louridas, P.: Using Wikis in Software Development. IEEE Software, 23, 2, 2006, pp. 88-91.

[MaWY06] Majchrzak, A.; Wagner, C.; Yates, D. 2006. Corporate wiki users: results of a survey. In: Proceedings of the 2006 International Symposium on Wikis, Odense, Denmark, ACM Press, New York, NY, 2006, pp. 99-104.

[Onto06] Ontoworld: Semantic Wikis. URL: http://ontoworld.org/wiki/Semantic_wiki, 2006.

[RBGH07] Rodriguez, F.; Berkling, K.; Geisser, M.; Hildenbrand, T.: Evaluating Collaboration Platforms for Offshore Software Development Scenarios. In: Meyer, Bertrand (eds): Proc. of the First Int. Conference on Software Engineering Approaches For Offshore and Outsourced Development (SEAFOOD'07), 2007, pp. 96-108.

[RaJa01] Ramesh, B.; Jarke, M.: Toward Reference Models for Requirements Traceability. In: IEEE Transactions on Software Engineering, IEEE Press, 2001, 27, pp. 58-93.

[RuEP03] Ruhe, G.; Eberlein, A.; Pfahl, D.: Trade-Off Analysis For Requirements Selection. In: Int. J. of Software Engineering and Knowledge Engineering, 13, 2003, pp. 345-366.

[SaVa01] Saaty, T. L.; Vargas, L.G.: Models, methods, concepts & applications of the analytic hierarchy process. Kluwer, 2001.

[Somm04] Sommerville, I.: Software Engineering. Addison-Wesley, 2004.

[TWIK06] TWiki.org: TWiki Success Stories. URL: http://twiki.org/cgi-bin/view/Main/TWikiSuccessStories (26.09.2006), 2006.

[VöKr06] Völkel, M.; Krötzsch M.; Vrandecic D.; Haller, H.; Studer, R.: Semantic Wikipedia. In: Proc. of the 15th Int. Conf. on World Wide Web, Edinburgh, May 23-26, 2006.

[W3C04] W3C 2004: The Web Ontology Language (OWL) Specification. URL: http://www.w3.org/TR/owl-features/ (16.09.2006), 2004.

A Visual Approach to Traceability and Rationale Management in Distributed Collaborative Software Development

T. Hildenbrand and A. Heinzl and M. Geisser and L. Klimpke and T. Acker
Department of Information Systems I
University of Mannheim, 68131 Mannheim, Germany
{thildenb; aheinzl; mgeisser; lklimpke; tacker}@rumms.uni-mannheim.de

Abstract: *Traceability* and *rationale management* are utmost critical, especially in distributed collaborative software development projects, due to a lack of mutual workplace awareness and informal coordination among the participating stakeholders. Therefore, this paper presents the rationale behind and implementation of a conceptional design for a novel approach to traceability and rationale management in distributed settings. In doing so, an innovative solution for extracting, visualizing, and analyzing the relationships between requirements and other key artifacts as well as responsible stakeholders is designed based on business needs within the software industry. Thus, the main objective of this paper is to instantiate the underlying conceptual considerations and methodological guidelines with the help of a comprehensive tool infrastructure particularly adapted to distributed settings.

1 Introduction

Traceability and *rationale management* represent utmost important tasks with respect to the governance of geographically distributed software projects [HGK07, dSHR07]. *Traceability*, in particular, denotes the ability to follow the relations between artifacts, such as requirements, design documents, or source code and responsible stakeholders [GF94]. *Traceability management* (TM) therefore involves activities including the identification, analysis, and maintenance of these relationships [KS98]. *Rationale management*, on the other hand, addresses the documentation and usage of rationale information regarding design and change decisions within software development projects [DP01]. In combination with the concept of *value-based software engineering* (VBSE), denoting that not all software artifacts can be attributed identical (customer) *value* and thus should be represented accordingly [EBHG05], enhancements to software project decision support can be achieved through the application of value-based and integrated end-to-end *traceability and rationale management* (TRM) covering the *entire* software development life cycle. Therefore, especially within spatially and temporally distributed projects, an increase of development efficiency and effectiveness can be achieved by providing (bi-directional) traceability information [dSHR07]. Moreover, an intuitive representation of the data by means of graphical visualization can be especially helpful and effective [dS05]. This quality feature is also required by the *Capability Maturity Model Integration* (CMMI) process

standard, for instance, as well as most other important (software) industry standards.

Therefore, the main *goal* of this paper is to document the design and implementation of a novel *tra*ce *vis*ualization approach called "TraVis", which extracts data related to different artifacts, activities, and users from collaborative software development environments in order to support visual analysis and collaborative management of the relations between these entities, while enhancing the platform's functionality by graphical visualization and analytic filtering mechanisms.

The following section 2 will, at first, briefly introduce the design and implementation *methodology* that was chosen for the design of the solution approach. After that, the main functional requirements for the novel solution will be outlined in section 3 before section 4 addresses the implementation platform and basic technologies used for realizing TraVis as part of a larger solution architecture. According to section 2, implementation details regarding the tool and application examples are given in section 5 to provide an initial evaluation in terms of feasibility of the proposed solution. The last section 6 provides a summary of findings and gives an outlook on future work.

2 Methodology

To eventually implement a comprehensive TRM solution, an *object-oriented* design methodology is chosen [Kru04], since trace relation models represent complex interrelated real-world objects such stakeholders and artifacts with different attributes such as descriptions, version numbers, and change rationale (see underlying information model in figure 1). The concurrent requirements management process is mainly driven by common TRM process activities and fields of application which in turn can be regarded as functional requirements or *"use cases"*[1] in the figurative sense guiding the further development process (see also table 1).

Moreover, this object-oriented approach aims at an architecture consisting of independent application *components*, thus, abiding by the main software engineering (SE) best practices formulated by [Kru04], i.e. (1) *iterative* development in combination with different evaluation steps and feedback loops to gain utmost utility of the novel approach [HMPR04], (2) use case-driven and *collaborative requirements management* involving several research groups and other stakeholders, as well as (3) designing a *component*-based solution architecture while separating three independent *system* components, namely (a) data model and persistence layer, (b) collaboration platform to support the overall TRM method, and (c) specialized tool support for traceability capturing, representation, and analysis (TraVis).

Before introducing the actual solution design and implementation, the following section summarizes the major functional requirements that have been gathered by means of several preceding studies.

[1] In this paper, the requirements will not be represented in the form of actual use cases as, for instance, defined by [Kru04] and [BME+07].

3 Functional Requirements and Related Work

In the following paragraphs, requirements for the novel solution elicited in preceding studies are presented in the form of distinct application areas—including a *procedural description* with respect to TRM process steps, actors, requirements *rationale and origin* [Kru04] as well as related work (see also [SZ04] and [Hil08]). Origins and rationale behind these fields of application are based on a broad analysis of literature review data [HRH07] as well as empirical requirements elicitation (cf. analysis results in [dSHR07] as well as [Hil08]).[2]

The main functional areas and requirements regarding TRM in distributed settings derived by means of literature reviews and tool evaluation have been complemented by conducting two observational case studies encoded MBL and VCI[3] (see table 1 and [dSHR07]). Table 1 presents an overview of the three main requirements categories: (1) change management in general (CM), (2) trace *capturing* and maintenance (TCM), and (3) trace representation and analysis. Besides common literature [HRH07, HRG+08], these findings are substantiated by the MBL and VCI studies [dSHR07].

Table 1: Overview of Requirements from Case Studies and Reviews (CM = change management; TCM = trace capturing and maintenance; TRA = trace representation and analysis)

Requirement	**Cat.**	**Source/Reference**
Automatic change notifications	CM	MBL and VCI studies
Workplace awareness support	CM	[DCAC03] and MBL study
Traceability information supply	CM	GSD study, literature review
Collaborative trace capturing	TCM	VCI study, literature review
Collaborative trace maintenance	TCM	VCI study, literature review
Rationale management support	TCM	VCI study, literature review
Alternative visual representations	TRA	VCI study, literature review
Predefined views and filters	TRA	VCI study
Adjacency graph analysis	TRA	VCI study, literature review

Based on these functional requirements, the main functionality supported by the TraVis solution is presented with respect to related work. The requirements elicited here correspond to the main areas of TRM, i.e. (1) *CM and impact analyses* as major use case for utilizing TRM information as well as the process of capturing and analyzing this information. However, in order to support CM in general, advanced methods for trace capturing (TCM) and analysis (TRA) are required as well. In addition to analyzing change impact, traceability information is also critical for (2) *project status reporting* and (3) *overall documentation of traces* by developers and other team members. This functionality can further be subdivided into (a) capturing and generation, (b) storage and representation, (c) analysis and maintenance use cases [Hil08]. In addition, this information is vital to project and

[2] For a more detailed requirements analysis and review of related work see [Hil08].

[3] Real names have been changed by the authors and these acronyms simply represent variable names without any semantic meaning.

program management for (4) *performance monitoring* purposes. Enhanced information supply can additionally assist (5) *post-specification development* tasks that will also be discussed in the following paragraphs.

Change Management and Impact Analysis. *Change management* (CM) tasks turn out to be the prevailing field of application for traceability information—e.g. for change propagation, generating notifications, and facilitating *impact analyses* in particular. Especially in the latter case, traceability information is critical in case of late changes for determining (1) directly and (2) indirectly affected artifacts and thus be able to (3) estimate the resulting *overall* costs of changes proposed in order to decide whether the change can be conducted or not [KS98]. Most often, change impact analysis pertains to changing *requirements* after an initial specification has been defined—e.g. in the form of change requests posted by different stakeholders [Som07]. These include the integration of new requirements as well as deleting and changing existing ones [Poh07].[4] The positive effects of sophisticated traceability information on impact analysis quality and efficiency has also been substantiated empirically (cp. for instance [LS96] and [LS98]). Automatically generated notifications as well as visual representations of dependencies can substantially support impact analyses [EH91].[5]

Project Status Reporting. CM also includes requirements implementation *status tracking and reporting*, i.e. capturing and analyzing the requirements' implementation status in post-specification project phases [Sch02]. In order to be able to determine and analyze the exact status of one particular requirement's implementation, *continuous horizontal traceability* from the *software requirements specification* (SRS) via architectural models, source code, and test cases must be established [Poh07]. This also requires information about contribution structures [Got95], i.e. authorship information [dSDR+04], and underlying rationale pertaining to post-specification artifacts and enables ensuring that all current project activities are based on actual customer demands and thus create customer value. Again, adequate quantitative data and according *visual* representations can be seen as possible approach to supporting this task. Moreover, visualizations not only facilitate inter-developer communication and project management, but also foster customer understanding and thus eventually system *acceptance*.[6]

Overall Project Documentation Support. The overall project documentation subsumes information necessary for both impact analysis and status reporting tasks. Taken together, a project's documentation corresponds to the overall *traceability network* containing both pre- and post-specification information with respect to artifact relations, design and change rationale, as well as contribution structures [KS98]. Project documentation tasks, therefore, pertain to the *entire* TRM process from capturing via storage and representation to

[4] Moreover, traceability information facilitates identifying cause and estimating the impact of bugs within the scope of software *maintenance* and *re-engineering* of legacy systems [Poh07].

[5] Current RM tools, such as DOORS and CaliberRM, do not provide full impact analysis support with respect to a complete analysis of all post-specification artifacts [GHR07].

[6] As it is the case for impact analysis, current RM tool do not allow a continuous post-SRS status tracking.

analysis and maintenance [Som07]. As has been argued before, disposing of relevant traceability information is vital to numerous other TRM-related tasks and can in turn facilitate distributed collaboration on the whole. Especially with respect to distributed development scenarios, current RM and SE solutions cannot provide integrated information and tool support [HRH07, GHR07, HRG+08].

Project Documentation Capturing and Generation. As regards *capturing* traceability network information, in particular, a *collaborative* approach based on one central repository is suggested both in literature and practice (cf. findings in [dSHR07]). Moreover, a common *metamodel*, including artifact entities, traces, and semantics as well as methodological guidelines and policies in the form of a traceability *manual* help coordinating this activity. Automatically *generated* suggestions for candidate links, on the other hand, can provide additional decision support, but are not considered here any further, since so far only research-in-progress solutions exist, which most often still require manual intervention and/or artifact description constraints [Hil08].

Project Documentation Storage and Representation. Based on the assumption of a central *storage* of traceability information, all distributed project stakeholders can be provided with adequate *representations* of relevant extracts retrieved by means of filtering and search techniques. Due to the complexity of large-scale distributed software projects *visualizations* facilitate stakeholder communication as compared to standard list and table representations. Currently, the information necessary for end-to-end TRM can only be provided by collaboration platforms [HRH07, HRG+08].

Project Documentation Analysis and Maintenance. With respect to TRM information *analysis and maintenance* support, filtering and search mechanisms need to be complemented by more advanced *visualization* and analysis methods such as adjacency graphs for more systematic impact analyses. However, current RM tools and collaboration platforms usually provide analysis functionality only based on matrix and linked list representations (cp. above and findings in [GHR07, HRG+08]).

Project Monitoring and Inspection. Within the scope of project monitoring activities, which are mostly conducted by project managers and other high-level stakeholders, the overall development process in terms of *who has done what and when* (process data) needs to be traceable at any given time [DP98]. On this basis, project managers have to be able to assess and report individual and team *performance*, *balance* the overall work load, and maintain reasonable *division of labor* while also considering implementation status reports (see above). To be able to do so, end-to-end traceability information is utilized to understand relations and particularly dependencies between artifacts and the stakeholders involved. Moreover, process data on tasks performed, resources consumed, and other quality measures can be utilized for general project planning and control [DP98].[7]

[7] Current development environments mainly focus on *source code* monitoring [HRG+08]. Other approaches, such as Ariadne [dS05], monitor socio-technical relations, but again only based on *source code* dependencies

Post-Specification Requirements Management. Requirements management tasks also comprise validation, verification, testing, and establishing standards compliance [SZ04]. Contribution structures, for instance, can be utilized to identify and involve relevant stakeholders into *validation* activities. As regards *verification*, refinement, dependency, and satisfiability relations allow for ensuring that all requirements specified have been allocated to ensuing implementation tasks and corresponding artifacts such as models and code. Similarly, traceability relations can be used to check the existence of appropriate test cases for verifying different requirements [SZ04] and to retrieve those.

Other Post-Specification Tasks. Traceability information and management capabilities can also support *other general SE tasks* such as finding the right stakeholders for *communication* and *coordination* purposes, artifact *understanding* and software *reuse* as well as software *maintenance* and thus support SE decisions due to better overview, visualizations, and analysis methods, e.g. for finding relevant information and/or contact persons more quickly. *Group or team awareness* is crucial in distributed settings due to the volatility of communication networks and partially sparse interactions among related stakeholders [HMFG00]. Appropriate visualization of relations between users and/or artifacts (cp. also [dS05]) can thus lead to more purposeful collaboration.

Artifact understanding, informed software reuse, and maintenance can also be accounted to general SE tasks that can benefit from traceability information. Improved traceability supports different stakeholders in *understanding* artifacts and their respective contexts even when not having contributed to their creation [SZ04]. For full artifact comprehension, *rationale* capturing, representation, and analysis capabilities are critical as well [RJ01]. Furthermore, requirements dependencies can support software *reuse* in that similar requirements are identified when the stated requirements are compared with existing requirements for indicating possibly reusable components from different artifact stages. In general, similarities between artifacts on different levels of horizontal abstraction along the software development life cycle can be utilized to manage application frameworks and software product lines [SZ04].

4 Implementation Platform and Technologies

The following sections briefly introduce the underlying information model implemented in the collaboration platform which the TraVis tool is based on. Moreover, other technologies used for the implementation of the TraVis solution approach and relevant implementation details are also outlined.

and code authorship information.

4.1 Traceability Information Model

Based on general trace information models, the underlying platform's inherent information model and its accessibility via the Web service API have to be adapted to TraVis. Compared to the underlying platform information model, some interrelations had to be simplified due to the vendor's practical restrictions. However, all vital elements of the TraVis information model are represented—*tracker items* and all different kinds of *artifacts* (documents, wiki pages, source code, etc.), for instance, can be tracked by distinct *realization states* and *versions* as well as categorized by embracing trackers or containers, respectively. *Rationale* information can be added by means of (wiki) comments to types of associations between tracker items and artifacts.

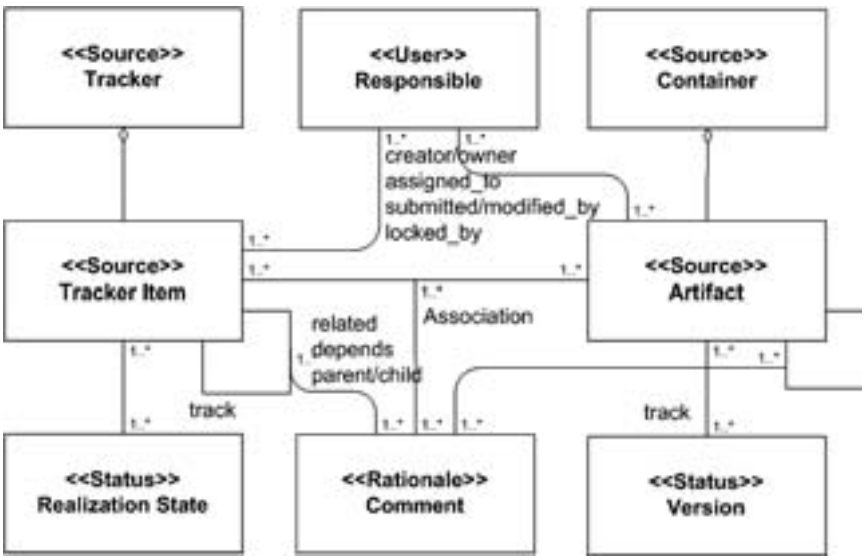

Figure 1: CodeBeamer Information Model

As can be seen in figure 1, the model differentiates between four basic types of *associations*: `depends`, `parent`, `child`, and `related`. Moreover, responsible *users* can take on various roles as they are associated to certain tracker items or artifacts. These include `owner`, `creator`, `assigned_to`, `submitted_by`, `modified_by`, and `locker` (someone who has locked a particular artifact for non-concurrent editing). *Change requests* are modeled as tracker items in a so-called change request tracker and therefore not represented separately in the CodeBeamer model. Furthermore, items in change request and requirements trackers also dispose of wiki-based rationale descriptions directly attached. For further adapting the CodeBeamer information model, specific project templates with predefined tracker structures are created. To be able to *connect* to the collaboration server and *extract* the relevant traceability information, TraVis uses CodeBeamer's Web service API (for a detailed description of the packages used see [HGK08]).

4.2 Implementation Technologies and Details

To be able to connect to the collaboration platform over the Internet and, thus, provide a Web-based user interface, the *Java WebStart*[8] (JWS) technology by Sun Microsystems is chosen. Moreover, JWS allows iterative updates to be able to extend the current prototype's functionality while keeping entailing network traffic low. This is and will be particularly critical for evaluating the overall solution with globally distributed stakeholders [Hil08].

For extracting the traceability information from the *collaboration platform*, the *Hessian*[9] *binary* Web service protocol provided by the underlying CodeBeamer platform is utilized. Besides its most advanced association mechanisms, link semantics, and wiki engine integration, the platform has initially been chosen for the prototypical TraVis implementation due to its fast and flexible Web service application programming interface (API) which has also been extended collaboratively with the vendor in the course of this research.[10]

The CodeBeamer platform provides an integrated *wiki engine* for (a) annotating and commenting on tracker items and other artifacts, e.g. to add *rationale* information, and (b) for creating self-contained wiki documents. Traceability information captured via wiki pages and comments can be in turn *represented* as interlinked wiki content in CodeBeamer's Web frontend and *analyzed* via the Web service API (see also [HGK08]). As for the WebStart application's user interface layer, TraVis uses *Java Universal Network/Graph*[11] (JUNG) framework. The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hypergraphs.

On basis of the different Web-based technology platform just described, the most important details pertaining to the implementation of TraVis are documented in figure 2. However, the focus of this paper is on the TraVis part of the overall solution architecture, i.e. visual representation, analysis, and maintenance functionality, and thus particular the use cases specified in section 3. The overall *solution implementation architecture* underlying this paper also includes an adapted collaboration platform as well as a separate source code management (SCM) repository (see figure 2). Moreover, particular semantic information can be added for more efficient retrieval of certain objects (cp. class descriptions in [Hil08, HGK08] and cf. information model in figure 1).

5 Tool Implementation and Application Scenarios

In this section, the functional areas presented in section 3 are substantiated to demonstrate how the underlying requirements are implemented by the TraVis solution and, thus, the

[8] `http://java.sun.com/products/javawebstart/` (2007-10-16).

[9] `http://hessian.caucho.com/` (2007-10-16).

[10] CodeBeamer has been analyzed and compared to other commercially available collaboration platforms, e.g. in [RGBH07], [HRH07], and [HRG+08].

[11] `http://jung.sourceforge.net/` (2007-10-16).

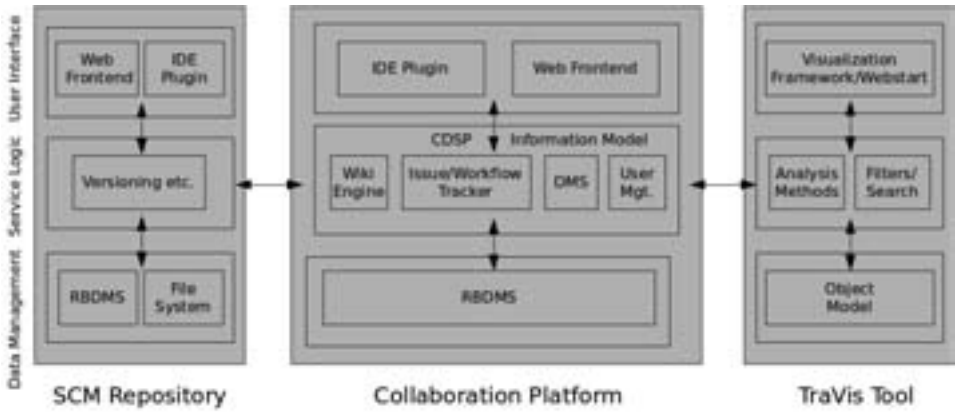

Figure 2: TraVis' Embedding Solution Architecture

feasibility of the approach as well as the applicability of the implementation is *evaluated* in terms of an initial demonstration or proof of concept. In doing so, TRM activities concerning (a) representation and visualization as well as (b) analysis and maintenance can be distinguished.[12]

5.1 Trace Representation and Visualization

TraVis provides different means of displaying the traceability network graph extracted from the development platform. Starting with an empty panel, the application's *user interface* requires the user to either manually select and deselect different element and association types or use various menu options for filtering, searching, and transforming the graph.

Manual Selection. The checkboxes of the TraVis *user interface* allow for a fine-grained configuration of the traceability information to be shown in the center panel. According to the CodeBeamer information model, the following elements are available for visualization: (1) *users* (stakeholders), (2) *issue trackers*, *tracker items*, and *attachments* (3) *documents* and *folders* (as parts of the document management system), (4) *forums* and single *posts*, (5) *wiki pages*, as well as (6) *source files*. When selecting particular project elements, only the resulting combination types of associations are activated while all other relations are shaded and not available. Accordingly, when removing certain information elements, these changes update the active options of manual selection. When checking or un-checking certain association types, these are added or removed, respectively. Checking

[12]The methodology for collaboratively *capturing* traceability information by means of the functionality incorporated in the underlying platform is not in the focus here, for further details see [Hil08]. Moreover, [Hil08] also documents three independent evaluation cycles and provides substantial evidence for the approach's *utility* in distributed scenarios.

Add/Remove all displays or hides all relation types possible. Moreover, *edge labels* can be manually added by means of the respective checkbox in the *Options* menu. Therefore, the checkboxes are the universal user interface for custom analyses concerning all major functional TRM requirements considered in this paper.

Element Filters. In addition to manually removing elements and associations from the graph, TraVis also provides numerous predefined filters for reducing network complexity and facilitating *inspection* tasks. To be able to do so, *inactive* users, other *disconnected* elements with no relations, as well as *closed* tracker items representing finished tasks can be filtered out automatically. Moreover, tracker items can be added to the graph, removed, and highlighted with respect to different tracker *categories*, implementation *phases*, and realization *states*. As has been mentioned earlier, TraVis complementarily analyzes links between wiki pages and thus these can be added or filtered out by checking the *Wiki Links* filter option. In addition to manually selecting and deselecting graph elements, filters are thus also universally applicable to a variety of TRM tasks such as status reporting and general traceability information management.

Predefined Views. Besides selection options and filters, TraVis also disposes of many other possible choices of graph representation. To facilitate overall usability and reduce complexity of the application, TraVis currently provides the following predefined views which can be easily adapted and extended to other use cases by means of TraVis' object-oriented and component-based architecture: (1) An "*Ego View*" with all elements associated to one particular stakeholder, (2) "*Editing (basic)*" and (3) "*Editing (all)*" showing elements that can be linked visually by TraVis, either regarding reduced or full project complexity, (4) "*Task Distribution*" with stakeholders and shared artifacts (see example below), (5) "*Major Artifacts*", (6) "*Tracker Structure*", as well as (7) "*Project Management Analysis*" which display tasks associated with certain stakeholders.

The *task distribution* view reveals who is doing what as well as *collaboration structures* formed by shared artifact relations between stakeholders, which is the basis for further analysis methods such as social network inspections (cp. next section as well as 6). In the case of this particular task distribution view, the implementation consists of four different graph options: (1) the *types of elements* and associations included (here tracker items, users, and their interrelations), (2) value-based *vertex sizing* (instead of uniform sizing, see subsequent section), (3) *mouse mode* (picking), and (4) graph *layout* (cp. also the following section). However, TraVis' architecture allows for adapting and creating views with more or less options very easily, e.g. by simply adding a new radio button in the *Views* menu.

Search Functions. For analyzing complex traceability networks, TraVis implements two complementary types of searches: (1) an integrated *type-ahead* search and (2) a *search menu* for adding and highlighting particular nodes. The former search function can be utilized to spot and find individual artifacts in very dense and complex graphs. In doing so, the type-ahead search already highlights graph items while the user is still typing, i.e. the

search process can be gradually concretized, while search results are instantly displayed in the center pane. For this and all other types of searching the traceability network information, the artifacts' titles and major description attributes are indexed and thus searchable. These search results are categorized according to the types of elements found—such as tracker items and documents in this case. By clicking particular elements in the result tree a graph can be built up from scratch or complemented (cp. also filtering mechanisms above). The *Find Artifact (Highlight)* function operates analogously to the type-ahead variant.

Transformations. TraVis also provides different graph transformations such as distortion, rotation and zooming. With respect to graph distortion, different lenses are defined with both hyperbolic and linear magnifying optics. To further reduce the graph's complexity, the different lens modes can be combined with all other options, filters, and views described so far. For zooming and rotating the graph both mouse and keyboard shortcut controls are provided in addition to the respective items in the *Options* menu of TraVis' user interface.

5.2 Trace Analysis and Maintenance

The implementation details described so far focused on visualizing the information retrieved from the platform for universally supporting different software engineering decisions and use cases. On top of that, TraVis also provides tool-supported methods for visual traceability network *analysis* and *maintenance*. Therefore, the following paragraphs explicate the TraVis functionality for exploring and analyzing the graph by means of additional information visualizations as well as visual editing capabilities.

Gradual Graph Exploration. Since the size of real-world traceability graphs are a major concern in TRM, *incremental exploration* techniques are a good solution for analyzing huge graphs originating from one particular element (cf. [HMM00], p. 37). This element can either be specified by a change request or found by the search and add function described above. The context menu option *Show connected vertices* adds all elements connected to a requirement resulting of a search operation, as well as the respective relation types as edge labels (`related`, `depends`, etc.). By applying this method in turn to one of the newly added elements, the graph is gradually explored from its origin. Furthermore, it is possible to show *all* connected vertices, i.e. the complete *adjacency* graph, of one particular start node by means of the corresponding option in the context menu (see figure 3). This type of information visualization therefore identifies artifacts directly and indirectly affected by changes to the focal artifact and thus facilitates *impact analyses* (cf. section 3). Moreover, *status reporting* is supported by enabling to follow the horizontal trace path of one requirement up to the current realization state. Additional artifact information is displayed on mouse-over operations in the form of tooltips. Vice versa, TraVis also allows for *removing* particular nodes.

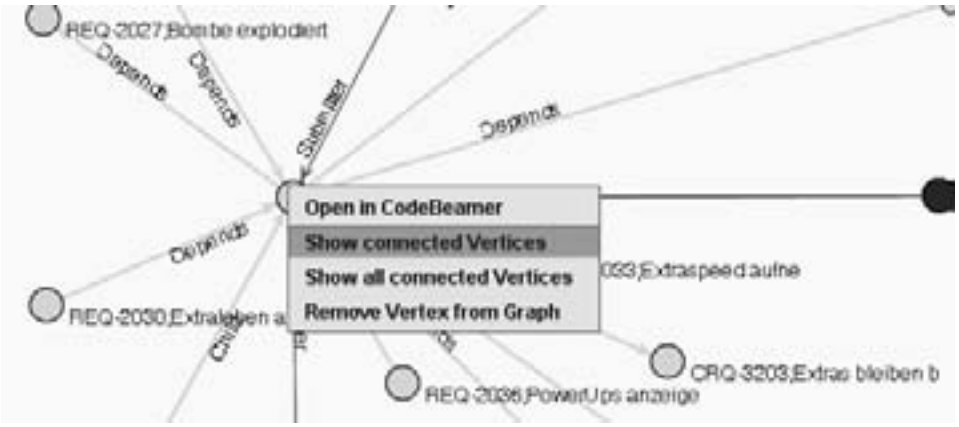

Figure 3: Gradual Graph Exploration Functionality

Trace Network Analysis. Besides manual trace network analysis by means of the checkboxes and options explicated above, TraVis also provides a comprehensive network analysis function activated by choosing the predefined *project management analysis* view. This non-visual "view" analyzes the number of traces among the different types of trackers and displays a detailed dynamically generated list of requirements and relations between tracker items. In doing so, customized trackers in addition to standard requirements, change requests, and bug trackers are considered as well. This function facilitates auditing the overall *project documentation* as well as determining traceability as a quality measure (cp. also data collection and measurement procedures in evaluation section. Again, the *show connected vertices* can also be applied to particular users and thus *monitored* what artifact development activities they are currently involved in. Using the *task distribution* view, on the other hand, also supports project monitoring and *controlling*—e.g. by spotting out project members that do not participate in any collaborative activities. Furthermore, the function *Team Statistics* in the *Project* menu returns a list of all project members' relations to certain project elements such as tracker items and documents, for instance, which also enables project managers to compare and assess the developers' collaboration intensity.[13]

Value-Based Node Sizing. Also mainly for *project management* (monitoring and controlling) purposes, TraVis implements variable node sizing algorithms for indicating customer value based on requirements analysis results such as priorities and other value measures. The customer value assigned to the requirements is then propagated to related and dependent artifacts such as design documents and source code—not including stakeholders. Therefore, the initial heuristic algorithm for calculating the node sizes has been improved by adapting the *PageRank* algorithm as also applied by the *Google*[14] search engine

[13] It has to be noted though, that tools such as TraVis can create a possibly unwanted form of transparency from the developers' perspective and raise data protection concerns that are beyond the scope of this research.

[14] `http://www.google.com/` (2007-10-20).

to the requirements of TraVis' solution architecture [PBMW98]. The PageRank algorithm is based on the assumption that nodes in a network, in this case artifacts and tracker items, are more relevant or *valuable* according to the number of references by incoming relations and their respective values. It has been shown that the values within a network converge after a finite number of iterations [PBMW98]. To be able to do so, TraVis initializes the nodes other than requirements with a value of $\frac{1}{number(nodes)}$ and defines a constant d[15] to accelerate convergence. The *new* value of one particular node is thus calculated as the sum of the related nodes' start values while the node's new *start* value is determined by $(1-d) + d * new\ value$.[16]

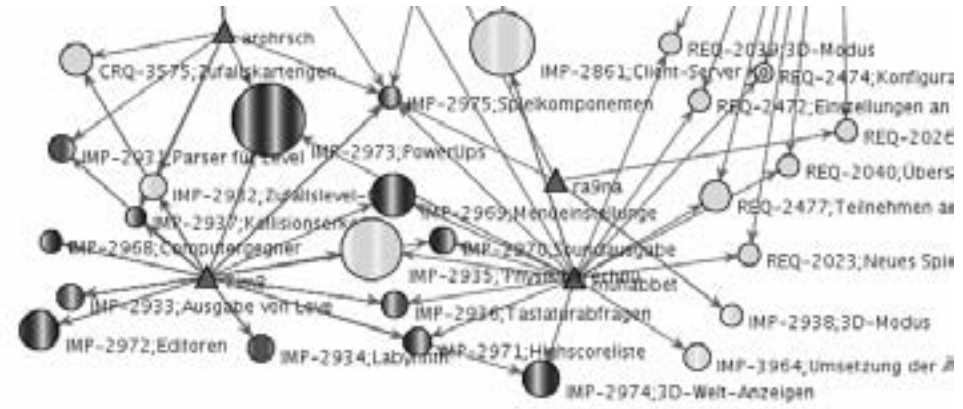

Figure 4: Extract from a Value-Based Task Distribution Graph

Value-based node sizing and the customer values written back to the platform as additional attributes, thus, provide decision support for prioritizing artifact-related activities according to their customer value [EBHG05] and, therefore, iterative as well as agile methods (see figure 4 for an extract of a value-based graph). Furthermore, customer value information also complements and quantifies the overall projects awareness as well as personnel turnover decisions (cf. section 3). Figure 4 also depicts how the different artifact types are encoded in different colors and patterns in order to provide even better visual information and decision support.

Rationale Information Management. Besides customer *value*, the artifact nodes of the traceability network carry a lot of additional information which can be utilized to comprehend justifications behind design as well as change decisions and version history—i.e. *rationale* information. To be able to prepare and provide this artifact context information

[15] Using a constant d is recommended by [PBMW98] and has been calibrated here to a value of 0.85 by means of the data from early evaluations and open source projects on the CodeBeamer-based JavaForge platform: `http://javaforge.com/` (2007-10-20). Currently, the TraVis implementation of PageRank converges after less than ten iterations for most projects. However, to provide some safety buffer, TraVis calculates 15 iterations.

[16] cf. [PBMW98] and [HGK08] for a more detailed description of the algorithm and calculation examples.

in an easily processable manner, the graphs are complemented by a so-called *element information* pane which displays a tree visualization of additional *artifact* or *stakeholder* information depending on the node currently selected in the center pane. While user element information is mainly useful for *monitoring* purposes, artifact rationale information facilitates both *impact analyses* and general *project documentation* and maintenance activities.[17]

Editing and Platform Synchronization. Starting with one of the predefined editing views (cp. section 5.1), for instance, or after manually selecting mouse mode *editing* in combination with any other view or filter, allows for removing and creating new associations between elements of the graph (see example in figure 5). This is conducted by simply dragging and dropping a line from one node to the other. As can be seen in figure 5, an association comment and type can be specified before the edge is added to the graph. Newly created edges as well as deleted ones are first recorded by TraVis' internal object model and later committed as a complete transaction to the platform by clicking on *Submit Changes* in the *Project* menu. Accordingly, changes made directly to the platform via its Web user interface can be synchronized by means of the *Reload Project* function. Visual editing essentially facilitates collaborative capturing and maintenance of traceability information and thus overall *project documentation* (cf. use case description in section 3).

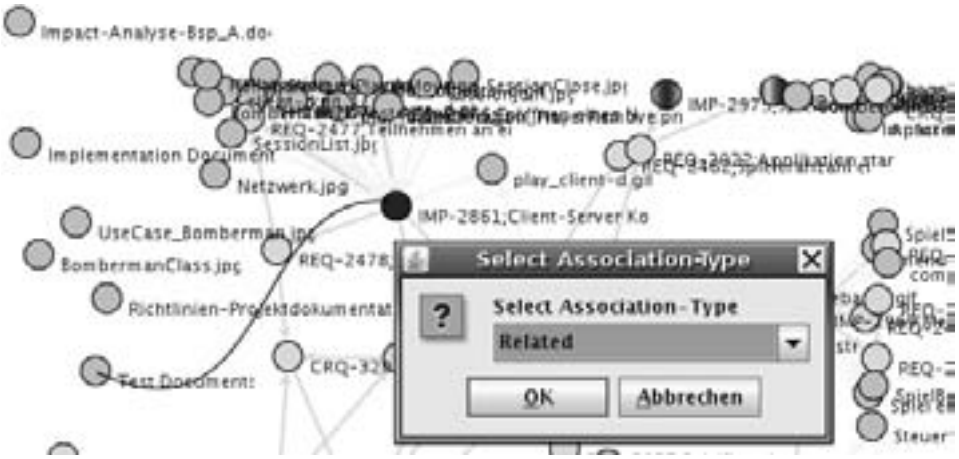

Figure 5: Visual Editing and Maintenance of Traceability Information

6 Summary and Outlook

As has been demonstrated in the preceding sections, the current prototype version implements all major functional requirements specified with respect to TraVis' *visual* rep-

[17] See [DMMP06] for further definitions of rationale management tasks in SE.

resentation, analysis, and maintenance support. It has also been shown that the adapted and customized version of CodeBeamer used for this prototypical implementation of the overall solution architecture covers the complementary *collaborative* capturing and maintenance processes. These TRM activities are particularly important in distributed software projects—mainly for enabling better mutual workplace awareness and informal coordination mechanisms due to the central availability of project knowledge in the form of traceability information.

Due to the fact that software projects can become very complex and spatial distribution of artifacts and actors hampers workplace awareness and process transparency, a lack of traceability can become harmful to project efficiency and documentation effectiveness in particular. Bidirectional end-to-end traceability, as required by CMMI, for instance, is critical, however, both with respect to in-project decision support and later maintenance tasks. Hence, TraVis provides added value particularly to distributed collaborative software development projects.

TRM methods as well as their combination with the VBSE approach can increase the quality of the artifacts developed within the individual project phases and, therefore, also the quality of the final product. For tool-based project support, the application of a software development platform that incorporates communication between all team members and enables information capturing should be considered. Additionally, tracking and managing particular requirements all the way to the new product are also supported [RGBH07].

Thus, within this paper, a novel trace visualization approach and respective tool support is introduced, which is based on such an underlying collaboration platform while enhancing the platform's functionality by graphical *visualization* as well as analytic filtering mechanisms and predefined views. In doing so, various TRM activities within distributed development projects are supported. Hereby, the focus of the tool's conception is mainly the support of cross-cutting project management functionality, requirements and change management in particular. This paper, thus, takes different requirements identified, substantiated, and verified in both, literature and real-world practice into account and implements a Web-enabled solution architecture based on an underlying central collaboration platform integrating various artifact repositories (cp. figure 2).

The latest version of TraVis described in this paper is complemented by enhanced *trace analysis* functionality and *predefined views* specifically designed to improve the tool's performance in certain TRM application respects [HGK07, Hil08]. Furthermore, for the *value-based* calculation of the size of the artifacts [HGK07], an advanced analysis method based on the *PageRank* algorithm is implemented [PBMW98]. Additionally, maintainability and usability of TraVis are further enhanced, for instance through additional filter and search functionality.[18] The main rationale for enhancing the solution with respect to *visual* and value-based analysis support is the need to reduce the complexity of relations in distributed reasonably and thus provide better decision support.

Within future conceptions and implementations of the tool support, functionality for addi-

[18] These enhancements are based on an initial set of two empirical evaluation studies concerning TraVis applicability in distributed settings [Hil08] and eventually aim at increasing the output quality and process efficiency of particular TRM tasks in distributed collaborative software development.

tional analyses like the graphical display of the *social networks* between project members and across multiple projects—both intra- and inter-organizationally—shall be integrated. For comprehensive social network analyses, *authorship* information pertaining to both pre- and post-specification artifacts is required [dSHR07]. Combining and visualizing information on artifact relation and contribution structures in turn allows for deriving social networks or sociograms[19] from socio-technical relations of artifacts and responsible stakeholders which provide a good basis for project and team structure analysis [dSRC+04]. This, in turn, can further facilitate team awareness, communication, and informal coordination [RvdHAA+07]. Personnel turnover situations that are most often noticeable in larger-scale projects, *offshore* outsourcing scenarios in particular, represent one major use case for this kind of information (see section 3 as well as [dSHR07]).

Besides these conceptional enhancements, already integrated functionality will be further evaluated experimentally to gather conclusions for future TRM requirements and development activities [Hil08]. In addition, several case studies and controlled experiments involving both students and partners within the software industry are planned in order to further evolve the solution and elicit remaining deficiencies as well as new requirements.

References

[BME+07] Grady Booch, Robert A. Maksimchuk, Michael W. Engel, Bobbi J. Young, Jim Conallen, and Kelli A. Houston. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, Boston, USA, 3rd edition, 2007.

[DCAC03] Daniela Damian, James Chisan, Polly Allen, and Brian Corrie. Awareness Meets Requirements Management: Awareness Needs in Global Software Development. In *Proceedings of the Workshop on Global Software Development (GSD'03)*, pages 7–11, 2003.

[DMMP06] Allen Henry Dutoit, Raymond McCall, Ivan Mistrik, and Barbara Paech, editors. *Rationale Management in Software Engineering*. Springer, Berlin. Deutschland, 2006.

[DP98] Ralf Dömges and Klaus Pohl. Adapting Traceability Environments to Project-Specific Needs. *Communications of the ACM*, 41(12):54–62, 1998.

[DP01] Allen H. Dutoit and Barbara Paech. Rationale Management in Software Engineering. In Chang SK, editor, *Handbook on Software Engineering and Knowledge Engineering*, volume 1. World Scientific, 2001.

[dS05] Cleidson R. B. de Souza. *On the Relationship between Software Dependencies and Coordination: Field Studies and Tool Support*. PhD thesis, Donald Bren School of Information and Computer Science, University of California, Irvine, USA, 2005. `http://www2.ufpa.br/cdesouza/pub/cdesouza-dissertation.pdf`.

[dSDR+04] Cleidson de Souza, Paul Dourish, David Redmiles, Stephen Quirk, and Erik Trainer. From Technical Dependencies to Social Dependencies. In *Proceedings of the Workshop on Social Networks at the 2004 International Conference on CSCW*, 2004.

[19] Sociograms are graph-based representation of social relations that a person has. In software projects, these can be derived from shared artifacts and communication structures [dS05].

[dSHR07] Cleidson R. B. de Souza, Tobias Hildenbrand, and David Redmiles. Towards Visualization and Analysis of Traceability Relationships in Distributed and Offshore Software Development Projects. In *Proceedings of the 1st International Conference on Software Engineering Approaches for Offshore and Outsourced Development (SEAFOOD'07)*. Springer, 2007.

[dSRC+04] Cleidson R. B. de Souza, David Redmiles, Li-Te Cheng, David Millen, and John Patterson. How a Good Software Practice Thwarts Collaboration: The Multiple Roles of APIs in Software Development. In *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'04)*, pages 221–230. ACM Press, 2004.

[EBHG05] Alexander Egyed, Stefan Biffl, Matthias Heindl, and Paul Grünbacher. A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, pages 2–7. ACM Press, 2005.

[EH91] Michael Edwards and Steven L. Howell. *A Methodology for Requirements Specification and Traceability for Large Real-Time Complex Systems*. Defense Technical Information Center, Fort Belvoir, USA, 1991.

[GF94] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proceedings of the 1st International Conference on Requirements Engineering (RE'94)*, pages 94–101. IEEE Computer Society, 1994.

[GHR07] Michael Geisser, Tobias Hildenbrand, and Norman Riegel. Evaluating the Applicability of Requirements Engineering Tools for Distributed Software Development. *Working Paper des Lehrstuhls für ABWL und Wirtschaftsinformatik der Universität Mannheim*, (2), 2007.

[Got95] Orlena Gotel. *Contribution Structures for Requirements Traceability*. PhD thesis, Department of Computing, Imperial College of Science, Technology, and Medicine, London, UK, London, UK, 1995.

[HGK07] Tobias Hildenbrand, Michael Geisser, and Lars Klimpke. Konzeption und Implementierung eines Werkzeugs für nutzenbasiertes Traceability- und Rationale-Management in verteilten Entwicklungsumgebungen. *Working Paper, Lehrstuhl für ABWL und Wirtschaftsinformatik der Universität Mannheim*, (5), 2007.

[HGK08] Tobias Hildenbrand, Michael Geisser, and Lars Klimpke. TraVis 2 - Ein Werkzeug für verteiltes, nutzenbasiertes Traceability- und Rationale Management. *Working Paper, Lehrstuhl für ABWL und Wirtschaftsinformatik der Universität Mannheim*, 2008.

[Hil08] Tobias Hildenbrand. *Improving Traceability in Distributed Collaborative Software Development—A Design-Science Approach*. Phd thesis, University of Mannheim, Germany, 2008.

[HMFG00] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. Distance, Dependencies, and Delay in a Global Collaboration. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, pages 319–328. ACM Press, 2000.

[HMM00] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[HRG+08] Tobias Hildenbrand, Franz Rothlauf, Michael Geisser, Armin Heinzl, and Thomas Kude. Approaches to Collaborative Software Development. In *Proceedings of the 2nd Workshop on Engineering Complex Distributed Systems (ECDS'08)*. IEEE Computer Society, 2008. accepted for publication.

[HRH07] Tobias Hildenbrand, Franz Rothlauf, and Armin Heinzl. Ansätze zur kollaborativen Softwareerstellung. *WIRTSCHAFTSINFORMATIK*, 49(Special Issue):S72–S80, 2007.

[Kru04] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, Boston, USA, 3rd edition, 2004.

[KS98] Gerald Kotonya and Ian Sommerville. *Requirements Engineering – Processes and Techniques*. John Wiley & Sons, Chichester, UK, 1998.

[LS96] Mikael Lindvall and Kristian Sandahl. Practical Implications of Traceability. *Software Practice and Experience*, 26(10):1161–1180, 1996.

[LS98] Mikael Lindvall and Kristian Sandahl. Traceability Aspects of Impact Analysis in Object-Oriented Systems. *Journal of Software Maintenance: Research and Practice*, 10(1):37–57, 1998.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. *Stanford Digital Library Technologies Project*, 1998.

[Poh07] Klaus Pohl. *Requirements Engineering – Grundlagen, Prinzipien, Techniken*. dpunkt.verlag, Heidelberg, Deutschland, 1st edition, 2007.

[RGBH07] Felix Rodriguez, Michael Geisser, Kay Berkling, and Tobias Hildenbrand. Evaluating Collaboration Platforms for Offshore Software Development Scenarios. In *Proceedings of the 1st International Conference on Software Engineering Approaches For Offshore and Outsourced Development (SEAFOOD'07)*, pages 96–108. Springer, 2007.

[RJ01] Balasubramaniam Ramesh and Matthias Jarke. Towards Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[RvdHAA+07] David Redmiles, André van der Hoek, Ban Al-Ani, Tobias Hildenbrand, Stephen Quirk, Anita Sarma, Roberto Silveira Silva Filho, Cleidson de Souza, and Erik Trainer. Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects. *WIRTSCHAFTSINFORMATIK*, 49(Sonderheft):S28–S38, 2007.

[Sch02] Bruno Schienmann. *Kontinuierliches Anforderungsmanagement: Prozesse, Techniken, Werkzeuge*. Addison-Wesley, Boston, USA, 2002.

[Som07] Ian Sommerville. *Software Engineering*. Addison-Wesley, Boston, USA, 8th edition, 2007.

[SZ04] George Spanoudakis and Andrea Zisman. Software Traceability: A Roadmap. In Shi-Kuo Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing, River Edge, USA, 2004.

The Emergence of Partnership Networks in the Enterprise Application Software Industry - An SME Perspective

Jens-M. Arndt, Thomas Kude, Jens Dibbern, and Armin Heinzl

jens.arndt, thomas.kude, dibbern, heinzl@uni-mannheim.de

Abstract: The enterprise application development industry is currently undergoing profound changes. The well established, large system developers (hubs) take the lead in establishing partner networks with much smaller, often young companies (spokes). This paper takes the perspective of these spokes and seeks to understand their motivations for entering into such partner networks. Drawing on research on strategic alliances and product complementarities, a theoretical framework on the determinants of partnering is developed. It is argued that partnering is especially attractive for smaller organizations when it allows them to access capabilities that would otherwise be difficult to obtain. Three broad categories of dynamic capabilities of hub organizations are assumed to act in this role: the capability to innovate architectures, the capability to provide an integrated enterprise application system, and the capability to address broad markets. These are analyzed in eight case studies. The cases represent small and medium sized enterprises (SMEs) that are participating in the partner network of a leading provider of enterprise application systems. The study reveals that while access to market capabilities is a key motivator for all spokes, the other two capabilities do not play an identical role in all cases. Rather, their impact on partnering motivation is contingent upon the type of solution offered by the partner in relation to the large system developer.

1 Introduction

The structure of the enterprise application software (EAS) industry has been subject to continuous change. Generally, EAS are information systems that reflect business-related organizational tasks and processes and the individual roles involved in these processes. The first of these EAS have been developed by hardware manufacturers. Then, during the 1960s, independent software vendors emerged that developed specific EAS for their customers. As more and more of these solutions had been used by companies, the systems integration effort increased significantly. As a consequence, in the 1980s, a trend towards consolidation started in the industry [CK03]. Offering comprehensive functionalities within one system avoided the need for complex, cross-solution integration. The enormous success of the providers of these integrated systems even reinforced the consolidation process [Dav98]. As a result, today the industry is clearly dominated by a few global companies and it has become common practice among customers to focus on a base system from one of these key players in the industry [Mer05]. This *jack-off-all-trades* strategy

[FMS98] is based on generic software. This implies that the basic functionalities provided by the integrated systems are industry best practice but not necessarily best practice for a particular customer [Som04]. The trade-off between generic and bespoke software has opened up the market for SMEs that provide very specific solutions for a small group of customers (*narrow specialist* strategy). Indeed, by integrating various specific solutions from different specialists into a coherent whole, customers may end up with superior systems [FSW00]. A more intense inter-organizational division of labor, which has been adopted in many other industries [BC00], has recently begun to play a more important role in the EAS industry. This has been fueled by the emergence of service-oriented architectures (SOA) which promise to reduce the integration effort between heterogeneous software applications [FSW00]. On the one hand, large vendors started to promote their systems with the ability to integrate third party solutions [Gre03]. On the other hand, SMEs have begun to realize that rather than fighting against windmills, it is more beneficial for them to cooperate with large systems providers [Ten03].

While certain types of cooperations such as joint ventures, strategic alliances, or licensing agreements have been observed in the industry for quite some time [RK94, GI06], more recently a new form of cooperation can be observed. Many of the large providers of integrated application systems have begun to actively foster partnership networks. These partnership networks may be described as loosely-coupled systems where the participants respond to changes in the partner's environment but still stay independent from each other [OW90] in that they are not linked by capital (joint venture) or through joint effort in a specific project or business area (strategic alliance), but by more general agreements which may be based on certifications of the other party's products or resources.

The underlying rationale for the emergence of these partnership networks has been rarely addressed in research. As a first attempt for closing this research gap, this study adopts the position of SMEs (spokes). The question is raised why these SMEs enter into partner relationships with large dominant players in the software industry. Drawing on the theory of *dynamic capabilities* for strategic alliances formation [ES96], three types of capabilities of hubs are identified [Hag93] that are proposed to be key motivators for smaller companies to become partner of a large systems provider [Ahu00]. The proposed reasons are combined in a theoretical framework and empirically examined in a multiple case-study design which focuses on one particular partnership network.

2 Theoretical Foundation

2.1 Networked Industry Structure

It has been argued above that currently a more intense division of labor is emerging in the EAS development industry. On the one hand, this division of labor implies that each company is focusing on its core competency, such as particular well defined software components [STT05]. On the other hand, it results in the necessity of a more intense cooperation between these specialized companies in order to ensure that particular entities can be

integrated into a coherent system. How such cooperation can be achieved has been widely studied. In general, two archetypes of cooperation are distinguished: The *completely intermeshed* and the *hub-and-spoke* network [SMC92]. In the first case, all companies are inter-connected with all others. Partners are dynamically selected as needed. In the second case, a core firm exists that inter-connects with all other organizations in a stable network. This central organization often takes on the role of a platform leader that is assumed to define technologies, markets, strategies, structures, and processes [GC02]. From these two organizational alternatives, the hub-and-spoke network closely resembles the above discussed structure that is currently emerging in the EAS development industry. The network is proposed to emerge around existing strong vendors and their systems (i.e., hubs), that form the network of partners (spokes).

In IS research, cooperative arrangements in the EAS industry have recently been studied more intensely. It has been argued that mergers and acquisitions [GI06] as well as strategic alliances [GI07] are formed in order to create value from complementarities that exist between different types of products and services. In the context of hub-and-spoke partnership networks, the hub was found to be responsible for developing the system's platform, which includes the general functionalities of standardized enterprise software. This platform is then complemented by the spokes, which are supposed to develop specific niche functionalities [GC02]. Through the existence of network externalities, the platform of such a central vendor becomes more valuable if more complementary products exist [SV99]. Thus, the attractiveness of taking on the role of this central platform architect increases with the growth of the network [MF93]. The question is raised, however, whether similar benefits from complementarities can also be realized by the spokes. Their ability to profit directly from network participation through externalities is limited. Thus, it has to be assumed that the key benefits originate from the dyadic relationship with the hub. In order to understand the underlying rationale for the spokes to enter into a partnership with a hub organization, it is essential to understand the unique capabilities that the hub brings into the network. While previous research has mostly focused on studying dynamic capabilities that spoke organizations have to develop on their own [MV07], this study argues that accessing the capabilities of the large partner also plays an important role for the spokes' competitiveness. Thus, in the following, the underlying theoretical foundations for the role of dynamic capabilities in partnership formation are introduced.

2.2 Access to Dynamic Capabilities as Inducement for Partnering

Previous research has predominantly drawn on the resource-based view (RBV) for understanding why organizations enter into cooperative relationships [IHV02]. By viewing firms as bundles of resources, it has been argued that the main reason why firms partner is to gain access to resources which they currently do not possess, but which the partner is offering [ES96]. This fact has also been labeled as the *duality* of inducements and opportunities [Ahu00]. In particular, *dynamic capabilities* are acting as inducements. They refer to the ability of using resources in a way that enables organizations to not only react to changes in their environment but to shape their environment to a certain extent

[TPS97]. This ability is particularly relevant in dynamic contexts, such as EAS development [MMS05]. For instance, the theoretical discussion will show that not only particular products or services are suggested to turn firms into attractive partners, but the capability to constantly invent new products and services and to bring them to market. In the following, we will discuss which dynamic capabilities are relevant in the context of hub-and-spoke partner networks in the EAS industry.

According to a large scale survey by [Hag93], network formation in high-tech industries such as EAS development, is motivated by three types of inducements: Speeding the process of innovation, accessing complementary technology, and gaining access to novel markets.[1] The capability to innovate has been found to be of prime importance for network formation [Fre91]. Historically, many periods of fast-paced technical progress can be explained through such a process of collective innovation across the boundaries of single firms [All83]. It has even been found that those organizations that emphasize innovations as a core part of their strategy are particularly inclined to enter into inter-organizational networks [ES96]. In a similar way, technology integration has often been stated as a prime motive for network formation. The rationale behind this argument lies in the increased complexity of modern technologies. As products have become increasingly complex, single organizations struggle in addressing the entire scope of their development. Thus, joint development of technology has been suggested in order to benefit from synergies between technological capabilities [ES96]. Finally, gaining access to certain markets has repeatedly been brought forward in the literature. This includes both the access to new markets through the augmentation of the product portfolio and the ability to address geographically remote markets [Hag93]. In the software industry, inter-organizational collaboration has been suggested as a key strategy for gaining access to marketing capabilities [RK94].

2.3 An Integrated Framework for Relationship Formation

Drawing on the theoretical insights from the formation of strategic alliances, a framework is developed that explains partnering from an SME perspective. As the discussion on the duality of inducements and opportunities has shown, these organizations can be assumed to enter into alliances because they have a specific need to integrate external resources [WW81]. Thus, the following section discusses capabilities that hub organizations possess and spokes lack. These are assumed to be the key motivating factors for the partnership formation from the spokes' perspective. They are developed based on the three general types of capabilities identified above.

[1] Although [Hag93] focuses on strategic alliances and thus on the joint deployment of resource, while this paper explicitly states that the here analyzed partnerships between hubs and spokes in the EAS industry usually do not imply such a joining of resources, [Hag93]'s taxonomy is still deemed a well suited starting point for building a research model.

Architectural Innovation Capabilities. Innovativeness plays a key role for organizations in high-tech industries such as the software industry, since they have to constantly cope with new technological advances as well as constantly changing customer requirements [Den04]. However, while innovativeness clearly constitutes one of the key dynamic capabilities of software firms [MV07], it is less clear how an SME benefits from the innovativeness of its large partner. More clarity is achieved through classifying innovations into different categories. As such, for industries that are characterized by a modular mode of operation, the distinction can be made between innovation at the component and the architectural level [HC90]. While component innovations accrue within the boundaries of one module, architectural innovations are affecting the way or the general structure by which the components are bound together to form a coherent system. A prominent and recent example for such an innovation that affects the assembling of different system components can be seen in the already mentioned emergence of SOA.

Architectural innovations require the capability to understand interdependencies between the different components as well as the functionality of the entire system [HC90]. Stated in other words, these architectural innovations are not confined to the narrowly circumscribed components in which the SMEs specialize. Thus, it can be assumed that SMEs face difficulties in developing innovations on the architectural level. Contrary, the capability to innovate on an architectural level is closely aligned with the business of integrating components or modules into systems which is exactly the core business of large hub organizations. Their core competency is to build and maintain integrated systems based on internally or externally developed components. In order to stay competitive in the systems market, they have to constantly innovate their architectural capabilities. Since the components of SMEs need to be aligned with architectural innovations in order to be compatible with other components, access to state-of the art architectural knowledge is crucial for them. It is important to stress that the spokes' prime motive for partnering with a hub is not the access to specific innovations, but to the hub's capability to innovate. Spokes do not actually use the hub's products, but aim at providing a module of an overall system that fulfills the changing customer requirements. This can be achieved by partnering with hub organizations, which is articulated in the following proposition:

> *Proposition I.* Small software producers (spokes) are partnering with large IS producers (hubs) in order to gain access to their capabilities to develop architectural innovations.

Integrated Systems Provision Capabilities. Closely related to the ability to innovate on the architectural level are those capabilities necessary to provide a technological base. In the context of EAS development, this technological base is considered to be the integrated system, which the hubs provide. The capabilities necessary to provide such a system represent the core competency of the hub organizations. As discussed above, the capabilities of hubs are historically rooted in the trend towards systems consolidation and developing comprehensive systems within the boundaries of a single firm [CK03]. The capabilities for developing such highly complex and integrated systems go far beyond what SMEs can provide. They are rooted in a profound understanding of various underlying technological disciplines and their interrelationships, an understanding of the entire system behavior

in terms of relevant parameters, the ability to design the entire system, the ability to design most key components of the system, and the ability to assemble component interfaces [Pre03]. As an example, the capabilities to provide such an integrated system served as the foundation of the success story of large providers of enterprise resource planning (ERP) software during the last two decades. They were the first that enabled a seamless integration of the entire information flows within an organization which was the foundation of their success story [Dav98].

As today the systems landscapes of large organizations are dominated by the solutions of these systems developers, the majority of SMEs in the EAS industry have realized that the success of their business critically depends on the inter-operability of their own solution with that of the large systems providers [Mer05]. In order to achieve this inter-operability of their own solution with the large systems, SMEs need to be well informed about the functionalities and interfaces of theses systems. By partnering with such a large system provider, the SMEs can facilitate their access to this kind of information. Furthermore, the inter-operability of the different solutions can be ascertained. This helps to reduce uncertainty on both the spokes and the hub side. This leads us to the following proposition:

> *Proposition T.* Small software producers (spokes) are partnering with large EAS producers (hubs) in order to gain access to their capabilities to provide an integrated system.

Notably, these capabilities to provide an integrated system refer to the exploitation of existing product architecture potential, which has been defined by the hub organization. This rather short term oriented "synchronic" capability differs from the long term capability to introduce incrementally or radically new systems architectures which has been referred to as "diachronic" systems integration capabilities [Pre03] (see Proposition I).

Market Access Capabilities. A third set of unique capabilities of the hubs that makes partnering with them attractive for spoke organizations is related to the sheer market power of the hubs. The products and services of SMEs in the software industry are of little value for most customers without being integrated into their existing systems landscape, which is largely dominated by the products of large systems providers. Thus, SMEs critically depend on the market access capabilities of the large providers and their willingness to consider the solutions of an SME as a complementary product in their solutions landscape. Many SMEs also cannot afford to make large investments into marketing activities which makes it particularly interesting for them to profit from the sophisticated marketing capabilities of large providers through a partnership arrangement [RK94].

Also, the strong market reputation of large systems providers can help SMEs reducing customer uncertainty about the quality and long term-reliability of their products and services. As it has been mentioned above, the quality of software is difficult to assess in advance. Through a partnership with a large vendor, SMEs can increase the level of trust in their solutions and their sustainability, in particular when the resources or products of the SMEs are officially accredited by the hub organization, e.g. through a certification [SB04]. The partnership agreement may substitute a direct quality assessment by signaling the trust-

worthiness of the SME to the market [Spe73]. Thus, the SMEs benefit from partnering with established large systems providers by leveraging their own market access through the large installed base of the hubs and by benefiting from the reputation of the hubs. The final proposition therefore reads as follows:

> *Proposition M.* Small software producers (spokes) are partnering with large EAS producers (hubs) in order to gain access to their capabilities to address broad markets.

2.4 Summary

The preceding discussion has evolved around the research objective of why small EAS development organizations are partnering with large system providers. As the underlying theoretical perspective on this issue has been that of dynamic capabilities, it has been argued that especially those capabilities of hubs are of prime importance that are difficult for the spokes to obtain. The discussion has, thus, focused on benefits of the inter-organizational approach in the EAS development industry and yielded the insight that three broad categories of benefits promise to be relevant in this context. As such, benefits from the hub's innovation capability, systems integration capability, and market access capability have been identified. The proposed relationships are illustrated in Figure 1.

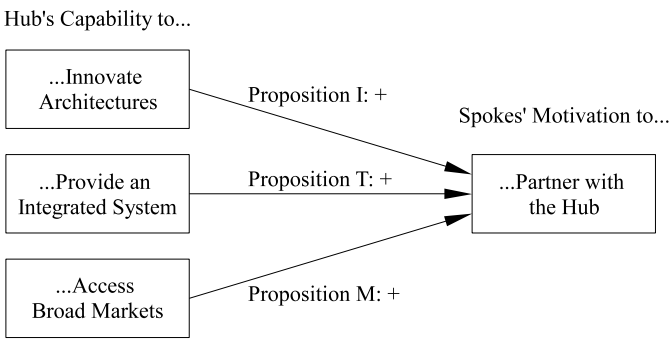

Figure 1: A Model for Explaining the Partnering Motives for SMEs in the EAS Industry.

3 Empirical Analysis

3.1 Methodology and Data Collection

According to [KL00], the choice of an appropriate research design is to a large extend determined by the research question that is intended to be answered. This paper deals

with the question *why* SMEs are partnering with well established large system providers. According to [Yin03], the case study approach is particularly promising to answer such *why* questions about motivations and rationales. The context in which case study research is especially well suited is characterized by two distinctive features. First, the boundaries between the studied phenomenon and its context are blurred. Second and closely related to this, a multitude of both variables of interest and available data covering these variables exist. Both features are clearly given in the above described context of EAS development. Obviously, various stakeholders and influencing factors are involved in this industry, and it is by no means clear which belong to the studied phenomenon and which are context.

Since this study is concerned with the motives of SMEs for partnering, the unit of analysis is the particular organization. Accordingly, a multiple-case study design was chosen [MH94]. It allows to investigate partnership formation by considering the contextual conditions of different organizations and, thereby, allows for an analytical generalization of the study findings [Yin03]. More specifically, generalization is achieved by applying literal replication logic, where each case is treated as a separate study for examining our proposed relationships [Yin03]. In order to enable the comparability of the individual cases, the focus was set on one particular partnership network which was established and is lead by one particular hub. This hub company is a large, global EAS vendor and one of key vendors for standardized solutions (known as enterprise resource planning (ERP) systems) discussed in the introduction of this paper. The recent version of the hub's EAS, however, is offered to customers as a platform based on SOA rather than a standardized, monolithic ERP solution.

Eight SMEs were selected that entered into a partnership with this hub organization. All analyzed case companies build on the hub's platform and, thereby, extend the overall system in a certain way. Each of the eight spoke companies is a certified partner of the hub, i.e., the hub has accredited that the solution of the spokes has the capability to technically integrate with the hub system. Table 1 introduces the case companies and their respective extensions of the overall system. It was ensured that each of the eight case companies represents an independent legal entity and is no subsidiary of any larger organization. Data from the spokes was collected from multiple sources, such as expert interviews, secondary material and personal observation between May and June 2007. Although the character of our analysis is rather exploratory, the expert interviews were guided by the propositions presented in section 2. As some of the case companies had even less than twenty employees, it was impossible to gain more than one interview partner for six out of the eight cases. For the other two cases, two and three interviews were conducted. On average, the interviews lasted one hour and resulted in a total verbatim transcript of 85 pages and more than 50,000 words of qualitative data[2].

For data analysis purpose, *codes* were developed for the three discussed propositions [MH94] by assigning a brief label for each of them: *Innovation*, *Technology*, and *Market*. Using this scheme, the transcripts of the interviews were then coded by assigning text passages to the three partnership motives proposed in the theoretical framework. These extracted interview fragments were then used for a two-stage analysis. First, a rough estimate

[2]Two interview partners did not give their approval to tape record the interview. Accordingly for these two interviews no verbatim transcript could be made. Rather, comprehensive notes have been taken by the authors.

Case Company A	Integration between the hub's system and various machines such as vending machines or intelligent refrigerators.
Case Company B	Integration between the hub's system and a CAD system of a different vendor.
Case Company C	Integration between the hub's system and a groupware system of a different vendor.
Case Company D	Providing systems for automatic, mobile data recording, used for example for inventory management.
Case Company E	Full-range supplier of IT systems and services for newspaper publishing companies.
Case Company F	Integration between the hub's system and various archiving systems.
Case Company G	Integration between the hub's system and various enterprise output systems, such as high-volume printers.
Case Company H	Providing a product information management system for cross-media publishing.

Table 1: The Analyzed Case Companies.

of the importance of each of the proposed benefits was assessed by counting the frequencies of the relevant fragments [MH94]. Then, a second round of analysis was conducted in which the underlying background of each fragment was carefully considered in light of each proposition [DWH07]. In the following, the findings from this two-stage process will be presented. Since space is limited, we will directly enter into the cross-case analysis. However, whenever necessary, the peculiarities of particular cases will be highlighted.

3.2 Data Analysis

Table 2 provides an overview of the number of relevant interview fragments for each of the proposed partnership motives. As can be inferred from this table, the hub's market access capabilities were the most frequently mentioned motive for entering into the partnership network. Indeed, in each case, this was named as the main motive. The second motive, in terms of frequency of quotes, was the hub's capability to provide an integrated system. Only rarely, the interview partner explicitly discussed the implications of the hub's capability to innovate systems architectures.

	Innovation	*Technology*	*Market*
Number of Quotes	3	13	26
Average per Interview	0.27	1.18	2.36

Table 2: Number of Relevant Interview Fragments.

A closer examination of the interview quotes largely confirmed the picture obtained from the frequency counting. Table 3 illustrates our findings of our qualitative data analysis for each of the three propositions and for each case. The proposed benefits for partnering are either supported (+), rejected (o), or even a reversed relationship could be found (-).

	Innovation	Technology	Market
Case Company A	o	+	+
Case Company B	-	+	+
Case Company C	o	+	+
Case Company D	o	o	+
Case Company E	o	o	+
Case Company F	o	+	+
Case Company G	o	+	+
Case Company H	o	o	+

Table 3: The Spokes' Reasons to Participate in the Network.

In all eight spoke cases, the interviewees unanimously declared that the primary reason for partnering was the expected benefit from the hub's market power. In this context, good support was found for the underlying rationale of this proposition. Essentially, all clients of the spoke companies were found to already possess a system developed by the hub organization. Thus, the value of the spokes' products was inevitably linked to the presence of the hub's system. Indeed, the integration interface of the spokes' solutions with that of the hub was seen as a main selling point for spokes. Through the partnership the spokes hoped to gain access to more customers that profit from the spokes' complementary functionality to the hub's system. In addition, the signaling aspect was also found to be of prime importance. The reputation that small companies gain from the partnering with a large, well recognized organization was confirmed to be crucial. Two spokes explicitly mentioned the fact that customers prefer their organization over competitors because of their partnership with the hub. Thus, clear support for the proposed benefit from the hub's market capability could be found in the collected data.

The examination of the data regarding benefits from the hub's systems provisioning capabilities as a driver for partnering has shown a more ambiguous picture. While for some case companies gaining access to information about functionalities and interfaces of the hub's systems were considered as very important (highlighted by a "+" in Table 3), others either did not even mention that aspect of the partnership or considered it of minor importance (highlighted by a "o" in Table 3). Notably, however, for none of the companies, the hub's capabilities to provide an integrated system were the driving force to enter into the partnership like it was found to be true for market access capabilities. Rather, for those companies that emphasized its importance, these capabilities of the hub were seen as a necessary precondition for realizing the final goal of market access that all spokes did have in common.

Even less support was found for the proposition on architectural innovativeness of the hub organization as a driving force for partnering. For none of the studied case companies,

access to architectural innovativeness was considered as a key factor for joining the partner network. Contrary to our proposition, the innovation capability of the hub was even seen as a potential threat rather than benefit in one particular case (Case B). Other spokes were unsure about the implications of architectural innovations by the hub.

Summing up, the data analysis revealed mixed support for our propositions. In particular, with regard to the benefits from systems provisioning capabilities of the hub, there are strong differences between two particular groups of organizations. These group differences call for a deeper investigation of the underlying reasons and potential theory refinement. This is addressed in the next section.

4 Discussion

When comparing the company profiles, as shown in Table 1, with the main differences between the case companies regarding Proposition T, as shown in Table 3, it becomes apparent that benefits from systems provisioning capabilities by the hub were only emphasized by those companies whose core business is the integration of other hardware or software components with that of the hub system (Cases A, B, C, F, G). In contrast, those companies whose core business is the development of software with business process functionality (Cases D, E, H), did not see the access to the hub's systems provisioning capability as a key benefit. Rather, they saw the inter-operability of their own system with that of the hub as a mere necessity. The integration itself was not seen as a key differentiator for their business model of selling business functionality systems. Accordingly, access to information about functionalities and interfaces of the hub's systems were not seen as providing any value per se. This is nicely illustrated by Case Company D, which develops a solution for automatic mobile data recording for voice controlled warehousing systems. This system allows for a more efficient inventory handling, even if it is not integrated with the partner's platform. Thus, although there is value in integrating the mobile recording system with the hub system by enhancing the efficiency of the inventory handling process through automatic rather than manual data transfer, the core value comes from the mobile recording functionality of the system. The same holds true for Case Company E. The solutions developed by this company are specialized on managing advertisement processes and transforming printed newspapers into online presence. Thus, the main added value for newspaper publishing companies is offered by the system's functionality. The integration with the hub's system, which allows the transfer of advertisement data directly into the standard business applications, only facilitates a better usage of the main solution functionality. The same story can be told for the cross-media publishing solution of Case Company H. Thus, taken together, the integration with the hub's system provides additional value for the customers of the spokes, but the main selling point is still the systems functionality, which is mostly unrelated to the inter-operability.

This completely deviates from the perspective of the second group. Their business model critically depends on their unique capability of providing interfaces between different technological infrastructures. This requires them to gain and maintain a profound understanding of the unique interface requirements of the two entities that they seek to integrate.

Since one of these entities is the system of the hub, the integrators see significant value in the information on functionalities and interfaces. The partnership makes it much easier for them to get access to this information which they need for their integration business.

Consequently, the impact of the hub's systems provisioning capabilities on the spokes' motivation to partner with the hub depends on the nature of the solution that the spokes provide. The key differentiator between the solutions is the focus of their business model. The spokes analyzed in our sample were found to either focus on solution integration or on (stand alone) business process functionality. Thus, each spoke can be placed on a continuum from low-high importance of business process functionality and low-high solution integration capability as depicted in Figure 2.

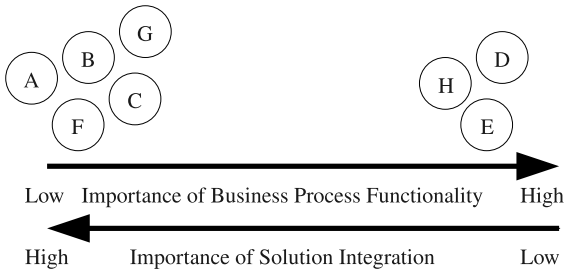

Figure 2: Importance of Business Functionality in the Partner's Solution.

Thus, refining the above sketched model, we introduce this importance of business functionalities as a moderating factor. Our data revealed that access to details about the provided system - especially regarding its interfaces - is of key importance to those partners that focus on integrating this system with another one (left side of Figure 2). In contrast, for those companies that focus on providing business process functionality (right side of Figure 2), this aspect is of minor relevance. This led us to the formulation of the following emergent proposition:

> Proposition BF_T: The higher the importance of business process functionality as opposed to solution integration for the business model of a spoke organization, the weaker is the proposed positive effect of the hub's capability to provide an integrated system on the spoke's motivation to enter into a partnership with this hub.

Taking the distinctive value proposition of the integrators into account, the fact that one of the integrators (Case Company B) actually saw the innovative capability of the hub as a potential threat rather than benefit becomes clearer. As far as the hub adopts a radically new systems architecture that makes the integration with external hardware or software components much easier for the hub, the business model to integrate this hub's system with another solution might be threatened. Thus, the architectural innovativeness of this specific hub might reduce its attractiveness as a partner. Consequently, the move towards more flexible component-based system architectures, such as SOA, may not be beneficial for all partners of the hub organization. Accordingly, Proposition I is modified as follows:

Proposition I_{ALT}. The architectural innovation capabilities of a large system developer (hub) are detaining small software companies (spokes) from partnering with this hub.

Notably, however, this reversed link is only proposed for spokes whose business model is based on solution integration. For SMEs that focus on providing systems with business process functionality the innovativeness of the hub plays a minor role for partnering. Accordingly, the following moderating impact is proposed:

Proposition BF_I: The higher the importance of business process functionality as opposed to solution integration for the business model of a spoke organization, the weaker is the proposed negative effect of the hub's architectural innovation capability on the spoke's motivation to enter into a partnership with this hub.

The enhanced theoretical framework is illustrated in Figure 3. The described innovation aspects are highlighted with dotted lines in this figure, since they are inherently exploratory in nature. The here collected data is not comprehensive enough to draw conclusions about their analytical generalizability. Thus, these links may viewed as theory emergent.

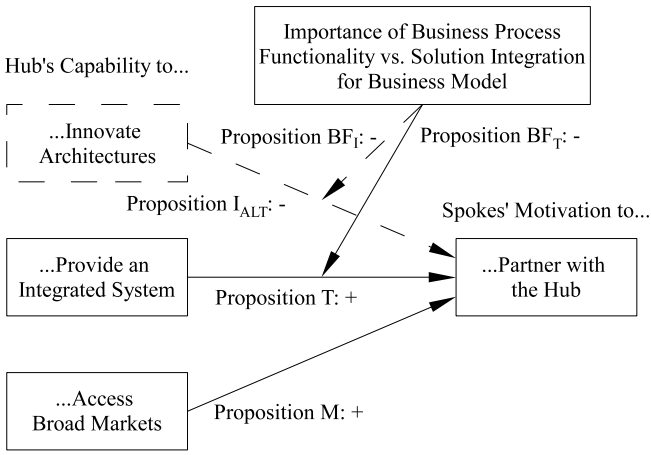

Figure 3: The Enhanced Model for Explaining the Motives of SMEs to Partner.

5 Conclusion

This paper has addressed the emergence of novel organizational structures in the EAS development industry. It builds on recent work that examined the role of complementarities for explaining inter-organizational forms of cooperation in the software industry [GI06], as well as the role of dynamic capabilities for SMEs in order to stay competitive in the software market [MV07]. The study is unique in that it focuses on new, more loosely-coupled forms of hub-and-spoke partnership arrangements that have emerged in the software industry. For explaining the emergence of these partnership networks, the concept of *duality* of inducements and opportunities for network formation [Ahu00] along with research on the motives of strategic alliance formation [Hag93] were applied. Benefits from three types of dynamic capabilities of large systems providers were proposed to motivate SMEs to enter into partnership networks of large systems providers. The resulting theoretical framework was empirically examined through a multiple case study including eight spoke companies that participate in the same partnership network of a large hub. Our findings revealed that the market access capabilities of the hub are a strong motivation for SMEs for joining the partnership network of the hub. Indeed, this was found to be the dominant motivation. A different picture emerged for the role of the hub's capabilities in architectural innovation and providing an integrated system. The impact of these two dynamic capabilities was found to be contingent on the type of solution that is provided by the spokes. In cases where the business model of spokes was focusing on providing integration interfaces between the hub and other hardware or software components, architectural innovation may even be seen as a threat rather than benefit, while access to the capabilities to provide an integrated system of the hub is seen as being important. In contrast, for spokes that develop solutions with business process functionality, both architectural innovativeness and systems integration capabilities of the hub played a minor role for partnering.

While it should be kept in mind that the findings from this research are based on a limited set of data and that the qualitative nature of our analysis may include some form of bias, we believe that our findings provide an interesting starting point for further research in the area of partnership networks in the software industry. One fruitful avenue for such an endeavor may be the application of the here developed framework from the hub perspective. This seems to be of prime importance, as it not only allows for a comprehensive analysis of the different roles in the network, but also of their interactions. Moreover, a closer analysis of the here developed propositions is promising. Especially a more distinct elaboration on the differences between groups of SME partners and their implications promises to be especially insightful not only for research, but for practitioners alike.

References

[Ahu00] G. Ahuja. The Duality of Collaboration: Inducements and Opportunities in the Formation of Interfirm Linkages. *Strategic Management Journal*, 21(3):317 – 343, 2000.

[All83] R. C. Allen. Collective Invention. *Journal of Economic Behavior and Organization*, 4(1):1 – 24, 1983.

[BC00] C. Y. Baldwin and K. B. Clark. *Design Rules - Volume 1. The Power of Modularity*. The MIT Press, Cambridge, USA, 2000.

[CK03] M. Campbell-Kelly. *From Airline Reservation to Sonic the Hedgehog: A History of the Software Industry*. The MIT Press, Cambridge, USA, 2003.

[Dav98] T. H. Davenport. Putting the Enterprise into the Enterprise System. *Harvard Business Review*, 76(4):121 – 131, 1998.

[Den04] P. J. Denning. The Social Life of Innovation. *Communications of the ACM*, 47(4):15 – 19, 2004.

[DWH07] J. Dibbern, J. Winkler, and A. Heinzl. Explaining Variations in Client Extra Costs Between Software Projects Offshored to India. *MIS Quarterly*, 31(Special Issue on IS Offshoring - forthcoming), 2007.

[ES96] K. M. Eisenhardt and C. B. Schoonhoven. Resource-Based View of Strategic Alliance Formation: Strategic and Social Effects in Entrepreneurial Firms. *Organization Science*, 7(2):136 – 150, 1996.

[FMS98] J. Farrell, H. K. Monroe, and G. Saloner. The Vertical Organization of Industry: System Competition versus Component Competition. *Journal of Economics and Management Strategy*, 7(2):143 – 182, 1998.

[Fre91] C. Freeman. Networks of Innovators: A Synthesis of Research Issues. *Research Policy*, 20(5):499 – 514, 1991.

[FSW00] M. Fan, J. Stallaert, and A. B. Whinston. The Adoption and Design Methodologies of Component-Based Enterprise Systems. *European Journal of Information Systems*, 9(1):25 – 35, 2000.

[GC02] A. Gawer and M. A. Cusumano. *Platform Leadership*. Harvard Business School Press, Cambridge, MA, 2002.

[GI06] L. S. Gao and B. Iyer. Analyzing Complementarities Using Software Stacks for Software Industry Acquisitions. *Journal of Management Information Systems*, 23(2):119 – 147, 2006.

[GI07] L. S. Gao and B. Iyer. Partnerships between Software Firms: Is There Value from Complementarities? Accessible online at: http://www.softwareecosystems.com, 2007.

[Gre03] J. Greenbaum. Build vs. Buy in the 21st Century. *Intelligent Enterprise*, 6(7):26 – 31, 2003.

[Hag93] J. Hagedoorn. Understanding the Rational of Strategic Technology Partnering: Interorganizational Modes of Cooperation and Sectoral Differences. *Strategic Management Journal*, 14(5):371 – 385, 1993.

[HC90] R. M. Henderson and K. B. Clark. Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly*, 35(1, Special Issue: Technology, Organizations, and Innovation):9 – 30, 1990.

[IHV02] R. D. Ireland, M. A. Hitt, and D. Vaidyanath. Alliance Management as a Source of Competitive Advantage. *Journal of Management*, 28(3):413 – 446, 2002.

[KL00] F. N. Kerlinger and H. B. Lee. *Foundations of Behavioral Research*. Harcourt College Publishers, Fort Worth, TX, 4th ed. edition, 2000.

[Mer05] P. Mertens. *Integrierte Informationsverarbeitung 1*. Gabler, Wiesbaden, GER, 15th edition, 2005.

[MF93] C. R. Morris and C. H. Ferguson. How Architecture Wins Technology Wars. *Harvard Business Review*, 71(2):86 – 96, 1993.

[MH94] M. B. Miles and A. M. Huberman. *Qualitative Data Analysis*. Sage Publications, Thousand Oaks, USA, 1994.

[MMS05] R. E. Miles, G. Miles, and C. C. Snow. *Collaborative Entrepreneurship - How Communities of Networked Firms Use Continuous Innovation to Create Economic Wealth*. Stanford Business Books, Stanford, CA, 2005.

[MV07] L. Mathiassen and A. M. Vainio. Dynamic Capabilities in Small Software Firms: A Sense-and-Respond Approach. *IEEE Transactions On Engineering Management*, 54(3):522– 538, 2007.

[OW90] J. D. Orton and K. E. Weick. Loosely Coupled Systems: A Reconceptualization. *Academy of Management Review*, 15(2):203 – 223, 1990.

[Pre03] A. Prencipe. Corporate Strategy and Systems Integration Capabilities: Managing Networks in Complex Systems Industries. In A. Prencipe, A. Davies, and M. Hobday, editors, *The Business of Systems Integration*, pages 114 – 132. Oxford University Press, Oxford, UK, 2003.

[RK94] P. M. Rao and J. A. Klein. Growing Importance of Marketing Strategies for the Software Industry. *Industrial Marketing Management*, 23(1):29 – 37, 1994.

[SB04] W. G. Sanders and S. Boivie. Sorting Things Out: Valuation of New Firms in Uncertain Markets. *Strategic Management Journal*, 25(2):167 – 186, 2004.

[SMC92] C. C. Snow, R. E. Miles, and H. J. Coleman. Managing 21st Century Network Organizations. *Organizational Dynamics*, 20(3):4 – 20, 1992.

[Som04] I. Sommerville. *Software Engineering*. Pearson Education Limited, Harlow, UK, 7th ed. edition, 2004.

[Spe73] M. Spence. Job Market Signaling. *The Quarterly Journal of Economics*, 87(3):355 – 374, 1973.

[STT05] N. Staudenmayer, M. Tripsas, and C. L. Tucci. Interfirm Modularity and Its Implications for Product Development. *Journal of Product Innovation Management*, 22(4):303 – 321, 2005.

[SV99] C. Shapiro and H. R. Varian. *Information Rules - A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston, MA, 1999.

[Ten03] B.-S. Teng. Collaborative Advantage of Strategic Alliances: Value Creation in the Value Net. *Journal of General Management*, 29(2):1 – 22, 2003.

[TPS97] D. J. Teece, G. Pisano, and A. Shuen. Dynamic Capabilities and Strategic Management. *Strategic Management Journal*, 18(7):509 – 533, 1997.

[WW81] J. A. Welsh and J. F. White. A Small Business is not a Little Big Business. *Harvard Business Review*, 59(4):18 – 27, 1981.

[Yin03] R. K. Yin. *Case Study Research - Design and Methods*, volume 5 of *Applied Social Research Methods Series*. Sage Publications, Thousand Oaks, USA, 3rd edition, 2003.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Busi-ness 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Erfahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometric and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für informatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^2CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering 2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de