Designing and Implementing a Tool for Distributed Collaborative Traceability and Rationale Management

Tobias Hildenbrand and Michael Geisser and Lars Klimpke and Thomas Acker
Department of Information Systems I
University of Mannheim, 68131 Mannheim, Germany
{thildenb; mgeisser; lklimpke; tacker}@rumms.uni-mannheim.de

Abstract: *Traceability* and *rationale management* are highly important—especially in distributed collaborative software development projects due to a lack of mutual awareness and informal coordination among the participating stakeholders. Therefore this papers presents a tool for extracting, visualizing, and analyzing the relationships between requirements and other artifacts, activities as well as users within a distributed software project using a collaborative development environment. Underlying requirements and the conceptual design of this tool are based on several real-world use cases demonstrating the respective value contribution of the tool's functionality.

1 Introduction

Traceability as well as *rationale management* are of significant importance within geographically distributed software projects (cp. [HGK07a] and [dSHR07]). Traceability, in particular, denotes to always be able to reconstruct which artifacts (for instance requirements, design documents or source code) are related to each other and how (cp. [GF94]). *Traceability management* (TM) therefore involves activities like the identification, analysis, and editing of these relationships [KS98]. *Rationale management* (RM), on the other hand, addresses the *documentation and usage* of rationale behind decisions within the software development process (cp. [DP01]). In combination with the concept of *value based software engineering* (VBSE), assuming that not all software artifacts generate the same (customer) *value* (see for instance [EBHG05]), enhancements can be achieved through the application of combined *traceability and rationale management* (TRM) throughout the entire software development process. Therefore, especially within spatially and temporally distributed projects, an increase of development efficiency and effectiveness can be achieved by upholding (bi-directional) traceability as a quality feature—as it is defined, for instance, within the CMMI standard. Especially regarding software change management and the associated impact analysis an intuitive representation of the data by means of graphical visualization can be helpful and effective.

The goal of this paper is thus to document the design and implementation of a trace visualization tool called "TraVis", which extracts data related to different artifacts, activities, and users from a collaborative software development platform in order to visualize and analyze the relationships between these entities. An initial version of the tool (cp. [HGK07a])

particularly emphasizes requirements and the resulting artifacts and the additional representation of rationale information. The second version of TraVis documented here is complemented by enhanced *trace analysis* functionality and predefined views specifically designed to improve the tool's performance in certain TRM use cases. Furthermore, for the *value-based* calculation of the size of the artifacts, an advanced analysis method based on the PageRank algorithm is implemented (cf. [PBMW98]). Additionally, maintainability and usability of TraVis are further enhanced, for instance through additional filter and search functionality. These enhancements eventually aim at increasing the output quality and process efficiency of particular TRM tasks in distributed collaborative software development. To clarify this, different tasks in form of use cases will be presented and referred to in this paper.

Section 2 will at first introduce the design methodology that was chosen for the implementation of the tool. Then specific requirements for the novel solution in form of use cases will be presented. In section 3, basic technologies used for implementing TraVis as well their concrete application within the software design are addressed. According to the use case defined in section 2, application examples are given in section 4 to provide a practical view and evaluation in terms of feasibility of the proposed solution. The last section provides a summary of findings and gives an outlook on future work.

2 Conceptional Design

This section introduces the design methodology applied for the implementation of the solution as well as in section 2.2 specific requirements in form of use cases.

2.1 Solution Design and Implementation Methodology

To eventually implement the use cases in form of a comprehensive solution, an *object-oriented* analysis and design methodology is chosen(cf. e.g. [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. The concurrent requirements management process is mainly driven by *use cases* which in turn relate to the common TRM process activities. This object-oriented approach aims at an architecture consisting of independent application *components*. Thus this work abides by the following main SE best practices (cf. also [Kru04]):

- *iterative* development (in combination with different evaluation steps),
- *use case*-driven requirements management (utilizing a collaboration platform), and
- *component*-based architecture (separating trace information model, collaboration platform, and actual visualization tool)

Iterative Development and Evaluation. To this end, this research applies a design science approach including multiple iterative evaluation cycles (cp. [HMPR04]). In doing so, the initial concept is first evaluated descriptively with respect to existing approaches and architectural fit, whereas the first prototype is applied in different real-world and university settings and access for beta testers and other stakeholders is granted.[1] Finally, an improved version of the solution is evaluated in controlled experimental settings in order to determine its usefulness and practicability in complex large-scale settings.[2]

Use Case-Driven Requirements Management. An iterative and evolutionary approach usually leads to many post-specification change requests and thus constantly changing requirements. To be able to manage the solution's continuous evolution and collaboration with different stakeholders such as beta testers and evaluation subjects, the main project is hosted on a commercial collaboration platform but open to anyone interested.

Component-Based Architecture. According to object-oriented design principles and patterns [Coa92], the overall solution is supposed to be component-based, i.e. consisting of units of independent deployment, possibly third-party composition, and with no externally observable states. As has been indicated before, the overall TRM approach consists of three main solution "components": trace information model, methodological guidelines, visualization and analysis component as well as a corresponding infrastructure. Within this approach, model and guidelines also have to be implemented by the tool infrastructure. Therefore, the *solution architecture* can be subdivided into three independent *system* components, namely (1) data model and persistence layer, (2) collaboration platform, and (3) specialized tools for traceability capturing, representation, and analysis (for general principles of object-oriented and component-based software design see [Som07]). Due to the fact that current design methodologies often lack focus on particular Web application issues in combination with support for componentization, more specialized methodologies for requirements analysis and high-level design applications are also taken into consideration for engineering the overall solution (these are among others [GM01], [LS04], and [BME+07]).

2.2 Traceability and Rationale Management Use Cases

In the following, requirements for the novel solution are presented in the form of distinct use cases—including a *procedural description* with respect to TRM process steps, actors, requirements *rationale and origin* as well as implementation *priority* (cp. also [Kru04] and [Wie05]). Rationale and origin of these use cases are mostly based on the analyses of literature review data (see [HRH07]) and empirical requirements engineering methods (cf. [dSHR07]), whereas requirements in real-world software projects are most often elicited by means of interviews and meetings (cf. [Som07]). These requirements can be allocated

[1] Beta testers were asked informally to test and comment on the different releases.

[2] Evaluation results will be published after finishing and analyzing *all* evaluation cycles

to the following more specific use cases most often mentioned in TRM literature and practice (cp. also [SZ04]). These represent different TRM-related tasks which are explicated in the following subsections.

2.2.1 Change Management and Impact Analysis

Change management tasks turn out to be the prevailing use case 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 (see [Som07], pp. 165). These include the integration of new requirements as well as deleting and changing existing ones ([Poh07], pp. 552). 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].

2.2.2 Project Status Reporting

Change management also includes requirements implementation *status tracking and reporting*, i.e. capturing and analyzing the requirements' implementation status in post-specification project phases (cp. [Sch02]). In order to be able to determine and analyze the exact status of one particular requirement's implementation, *continuous horizontal traceability* from the SRS via architectural models, source code, and test cases must be established (cf. also [Poh07], p. 504). This also requires information about contribution structures 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*.

2.2.3 Project Documentation Support

The project documentation subsumes information necessary for both impact analysis and status reporting tasks. Fundamentally, a project's documentation corresponds to the overall *traceability network* containing both pre and post-specification information with respect to artifact relations, rationale, and contribution structures (cf. [KS98]). Project documentation tasks therefore pertain to the *entire* TRM process from capturing via storage and representation to analysis and maintenance. 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. As regards *capturing* traceability network information, 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. Based on the central *storage* of traceability information, all 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. Integration with modern collaboration tools known from Web 2.0 contexts such as blogs, wikis, and general collaborative tagging capabilities can be seen as possible approach to support this task even more sophistically. With respect to *analysis and maintenance* support, filters and search mechanisms need to be complemented by more advanced visualization and analysis methods such as adjacency graphs for more systematic impact (cf. section 2.2.1) and social network analyses (SNA). For comprehensive SNA analyses, authorship information pertaining to both pre and post specification artifacts is required. This in turn can facilitate team awareness and communication as well as personnel turnover use cases (see section 2.2.5).[3]

2.2.4 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. 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* complementary to implementation status reports (see above) and despite the fact that developers and other contributers are possibly distributed. To be able to do so, traceability information is utilized to understand relations and particularly dependencies between artifacts and the stakeholders involved. Combining and visualizing information on artifact relation and contribution structures in turn allows for deriving social networks or sociograms[4] from socio-technical relations of artifacts and responsible stakeholders which provide a good basis for project and team structure analysis [dSRC+04]. Moreover, process data on tasks performed, resources consumed, and other quality measures can be utilized for general project planning and control.

2.2.5 Post-Specification Software Engineering Support

Requirements management tasks furthermore comprise validation, verification, testing, and establishing standards compliance (cf. [SZ04]). Contribution structures, for instance,

[3] SNA capabilities are in the center of further development activities as part of future work, see also section 5.

[4] 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.

can be utilized to identify and involve relevant stakeholders into *validation* activities. As regards *verification*, refinement, dependency, and satisfiability relations allow for determining all requirements specified have been allocated to ensuing implementation tasks and 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], p. 418) and to retrieve those.

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. Another general problem in software projects is *personnel turnover* management (cf. [Mob82]).

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], p. 420). To be able to do so, traceability relations between source code and manual pages, for instance, sometimes need to be reconstructed automatically [MM01]. 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. In general, similarities between artifacts on different levels of horizontal abstraction along the software development lifecycle can be utilized to manage application frameworks and software product lines (see e.g. [SZ04]). Moreover, traceability information facilitates identifying cause and estimating the impact of bugs within the scope of software *maintenance* and *re-engineering* of legacy systems (see [Poh07], p. 504).

3 Implementation

The following section briefly introduces the technologies used for the implementation of the TraVis solution and subsequently, relevant implementation details .

3.1 Implementation Technologies

To be able to connect to the collaboration platform over the Internet and thus provide a Web-based user interface, the *Java WebStart*[5] (JWS) technology by Sun Microsystems is chosen. For extracting the traceability information from the collaboration platform, the *Hessian*[6] *binary* web service protocol is utilized. Moreover, the underlying CodeBeamer platform has been analyzed and compared to other commercially available collaboration platforms (cf. [RGBH07]). Besides its most advanced association mechanisms, link semantics, and wiki engine integration, the platform is chosen for the prototypical TraVis implementation due to its fast and flexible Hessian Web service API which has also been extended collaboratively with the vendor in the course of this research.

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 [HGK07b]). The for the WebStart application's user interface layer, TraVis uses *Java Universal Network/Graph*[7] (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.

3.2 Solution Implementation Details

On basis of the different Web-based technologies just described, the most important details pertaining to the implementation of TraVis are documented in the following. In doing so, the focus 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 2.2. The *solution implementation architecture* assumed in this paper also consists of an adapted collaboration platform underlying TraVis. Therefore, this section presents the essential details of the adaptations and the data transfer process between the platform and TraVis.

CodeBeamer Information Model As already mentioned in section 3.1, the CodeBeamer platform has been adapted and extended in the course of this research. Based on general trace information models, CodeBeamer's inherent information model and its accessability 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,

[5] `http://java.sun.com/products/javawebstart/` (2007-10-16)
[6] `http://hessian.caucho.com/` (2007-10-16)
[7] `http://jung.sourceforge.net/` (2007-10-16)

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. 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 [HGK07b]).

Data Capturing and Object Model Implementation After the connection has been established, the user is promted to select a particular project. Choosing a project initiates the data capturing and synchronization process. In doing so, the data is captured from CodeBeamer's information model by means requesting the different types of Data Tranfer Objects (DTOs) over the Hessian Web service protocol. The traceability information captured is then stored in TraVis' internal object model which is implemented as a doubly linked list (see above). Altogether, TraVis utilizes three different kinds of lists: (1) `CbVertexList` for storing nodes, (2) `CbEdgesList` for representing edges, as well as (3) `CbAssocationsList` as auxiliary list which all inherit their behavior from `AbstractList`. Each list elements contains both a CodeBeamer DTO object as well as the respective JUNG framework object, e.g. implementing the `Vertex` interface. Moreover, particular semantic information is added for more efficient retrieval of certain objects (cp. class descriptions in [HGK07b]).

4 Use Case Application Examples

In this section, the use cases defined in section 2.2 are utilized to demonstrate how the underlying requirements are implemented by the TraVis solution's distinct functionalities and thus *evaluate* the feasibility of the approach as well as the applicability of the implementation. In doing so, TRM acitivities concerning (a) representation and visualization as well as (b) analysis and maintenance can be distinguished.

4.1 Trace Representation and Visualization

Once the traceability information is transferred to TraVis, different means of displaying the traceability network graph are provided. Starting with an empty panel, the application 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: (1) *users* (stakeholders), (2) issue *trackers*, tracker *items*, and *attachments* (3) *documents* and *folders* (as parts of the DMS), (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 in grey. Accordingly, when removing certain information elements, these changes update the active options of manual selection. When checking or unchecking certain association types these are added or removed, respectively. Checking *Add/Remove all* displays or hides all relation types possible. Moreover, *edge labels* can be manually adjoined by means the respective checkbox in the *Options* menu. Therefore, the checkboxes are the universal tool for custom analyses concerning all major TRM use cases.

Element Filters. In addition to manually removing elements and associations from the graph, TraVis also provides numerous predefined filters for reducing network complexity and facilitate *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 complemtarily 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) *Ego View*, (2) *Editing (basic)*, (3) *Editing (all)*, (4) *Task Distribution*, (5) *Major Artifacts*, (6) *Tracker Structure*, and (7) *Project Management Analysis*. 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 5). 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. 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 contents in the center pane instantly displays the search results. 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.

The functionality described above pertains to all major use cases in section 2.2. More use case-specific features are presented in the following subsection.

4.2 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 1). 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 2.2.1). 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.

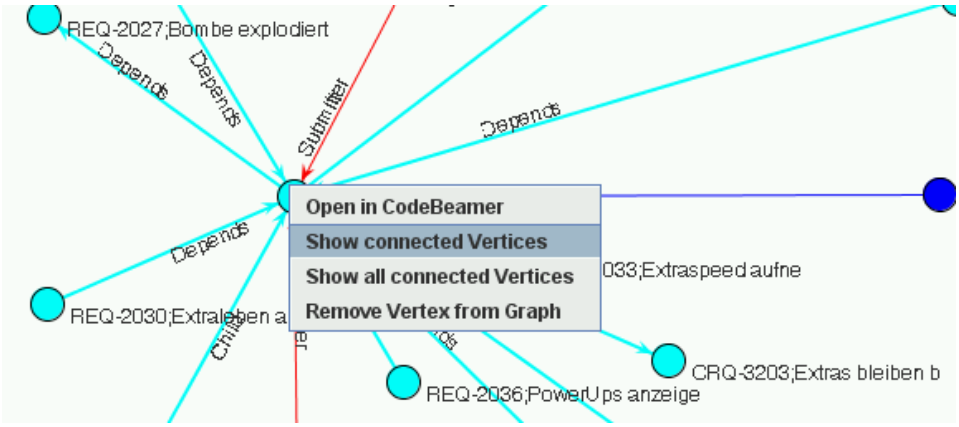

Figure 1: Gradual Graph Exploration Functionality

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.

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 (stakeholder nodes excluded). Therefore, the initial heuristic algorithm for calculating the node sizes has been improved by adapting the *PageRank* algorithm known from the *Google*[8] search engine to the premises of the solution architecture (cp. also [PBMW98]). The PageRank algorithm is based on the assumption that nodes in a network (here: artifacts and tracker items) are more important (or valuable) according to the number of 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 achieve this state, TraVis initializes the nodes other than requirements with a value of $\frac{1}{number(nodes)}$ and defines a constant d[9] of 0.85 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 $(1-d)+d*new\ value$. 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 and therefore iterative as well as agile methods (see 2 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 2.2.5).

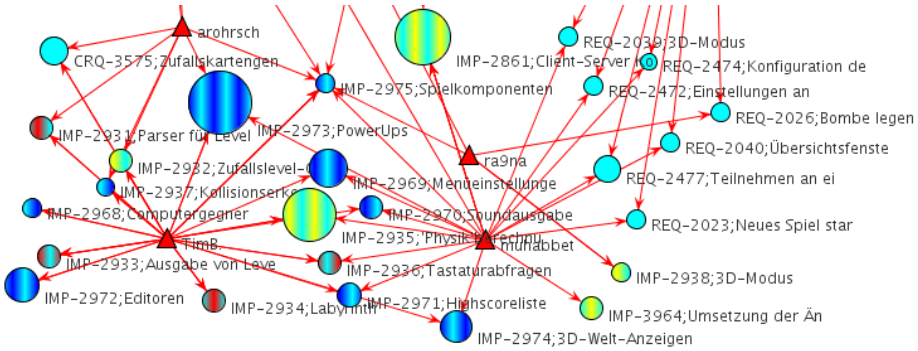

Figure 2: Extract from a Value-Based Task Distribution Graph

[8] `http://www.google.com/` (2007-10-20)

[9] Using a constant d is recommended by [PBMW98] and has been calibrated here by means of the data from early evaluations and open source projects on the CodeBeamer-based JavaForge platform: `http://javaforge.com/` (2007-10-20). The TraVis implementation of PageRank now converges after less than 10 iterations. However, to provide some safety buffer TraVis calculates 15 iterations.

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 and change decisions—i.e. *rationale* information. To be able to prepare and provide this artifact context information in an easily processible manner, the graphs are complemented by a so-called *element information* pane. Depending on the node currently selected in the center pane either additional *artifact* or *stakeholder* information is displayed. While user element information is mainly useful for *monitoring* purposes, artifact rationale information facilitates both *impact analyses* and general *project documentation* and maintenance activities.

Editing and Platform Synchronization Starting with one of the predefined editing views (cp. section 4.1), for instance, or after manually selecting mouse mode *editing* on the lower righthand side in combination with any other view or filter, allows for removing and creating new associations between elements of the graph. This is conducted by simply dragging and dropping a line from one node to the other. After that, 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 2.2.3).

5 Summary and Outlook

As has been demonstrated in the preceding sections, the current TraVis prototype implements all major functions and use cases specified with respect to TraVis' visual representation, analysis, and maintenance support. It has also been shown that the adapted version of CodeBeamer used for this prototypical implementation of the overall solution approach covers the complementary collaborative capturing and maintenance processes. These TRM activities are particularly important in distributed software projects—mainly for enabling workplace awareness and informal coordination. TRM methods as well as their combination with the VBSE approach can increase the quality of the artifacts to be developed within the individual project phases and therefore the quality of the final product. For tool-based project support the application of a software development platform that supports communication between team members and enables the storage of data should be considered. Additionally, tracking and managing requirements to the new product should also be supported (cp. [RGBH07]).

Therefore, within this paper, the novel trace visualization tool TraVis 2 is introduced, which is based on an underlying collaboration platform—here CodeBeamer, for instance—enhancing its functionality by graphical visualization as well as filtering. Through the visualization of the different relations between the artifacts and stakeholders as well as other

functionalities provided by the tool, various use cases that occur within a distributed development project are supported. Hereby, the focus of the tool is mainly the support of project management functionality—requirments and change management in particular. This paper documents how different use cases identified and substantiated in both literature and practice can be supported and implemented by an Internet-enabled solution architecture based on an underlying collaboration platform.

Within future versions of the tool, functionality for additional analyses like the graphical display of the social networks between project members and across multiple projects—both intra- and inter-organizationally—shall be integrated. The alternative visualization of the traceability network, e.g. a matrix view of artifact types that is already supported within RequisitePro (IBM), or a process view of artifacts, i.e. a classification of artifacts regarding individual process phases. Besides these enhancements, already integrated functionality should also be evaluated experimentally to gather conclusions for future TRM requirements and development activities. Moreover, additional case studies and controlled experiments involving partners within the software industry are planned.

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, 3. auflage edition, 2007.

[Coa92] Peter Coad. Object-Oriented Patterns. *Communications of the ACM*, 35(9):152–159, 1992.

[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`.

[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.

[GM01] Arthula Ginige and San Murugesan. The Essence of Web Engineering - Managing the Diversity and Complexity of Web Application Development. *IEEE Multimedia*, 8(2):22–25, 2001.

[HGK07a] Tobias Hildenbrand, Michael Geisser, and Lars Klimpke. Konzeption und Implementierung eines Werkzeugs für nutzenbasiertes Traceability- und Rationale-Management in verteilten Entwicklungsumgebungen. *Working Paper des Lehrstuhls für ABWL und Wirtschaftsinformatik der Universität Mannheim*, (5), 2007.

[HGK07b] Tobias Hildenbrand, Michael Geisser, and Lars Klimpke. TraVis 2 - Ein Werkzeug für verteiltes, nutzenbasiertes Traceability- und Rationale Management. *Working Paper des Lehrstuhls für ABWL und Wirtschaftsinformatik der Universität Mannheim*, 2007.

[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.

[HRH07] Tobias Hildenbrand, Franz Rothlauf, and Armin Heinzl. Ansätze zur kollaborativen Softwareerstellung. *WIRTSCHAFTSINFORMATIK*, 49(Sonderheft):S72–S80, 2007.

[Kru04] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, Boston, USA, 3. auflage 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.

[LS04] Seung C. Lee and Ashraf I. Shirani. A Component Based Methodology for Web Application Development. *Journal of Systems and Software*, 71(1-2):177–187, 2004.

[MM01] Jonathan I. Maletic and Andrian Marcus. Supporting Program Comprehension Using Semantic and Structural Information. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, pages 113–122. IEEE Computer Society, 2001.

[Mob82] William Hodges Mobley. *Employee Turnover: Causes, Consequences and Control*. Addison-Wesley, Reading, USA, 1982.

[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, 1. auflage 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.

[Sch02] Bruno Schienmann. *Kontinuierliches Anforderungsmanagement: Prozesse, Techniken, Werkzeuge*. Addison-Wesley, Boston, USA, 2002.

[Som07] Ian Sommerville. *Software Engineering*. Addison-Wesley, Boston, USA, 8. auflage 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.

[Wie05] Karl E. Wiegers. *More About Software Requirements: Thorny Issues and Practical Advice*. Microsoft Press, Redmond, USA, 2005.