Konzeption und Implementierung eines Werkzeugs für nutzenbasiertes Traceability- und Rationale-Management in verteilten Entwicklungsumgebungen

Tobias Hildenbrand and Michael Geisser and Lars Klimpke

Working Paper 5/2007
February 2007

Working Papers in Information Systems

University of Mannheim
Department of Information Systems 1
D-68131 Mannheim/Germany
Phone +49 621 1811691, Fax +49 621 1811692
E-Mail: wifo1@uni-mannheim.de
Internet: `http://www.bwl.uni-mannheim.de/wifo1`

1 Einleitung

1.1 Problemfeld

Unter Nachvollziehbarkeit (engl: Traceability) wird die Möglichkeit verstanden, verschiedene Artefakte (beispielsweise Anforderungen, Designdokumente (z.B. UML-Diagramme) oder Quellcode) so miteinander zu verknüpfen, dass stets nachvollzogen werden kann, welche Artefakte wie miteinander in Beziehung stehen. Die Nachvollziehbarkeit sollte hierbei sowohl vorwärts- als rückwärtsgerichtet möglich sein.

Ebenso wichtig wie die Nachvollziehbarkeit der Beziehungen zwischen den Artefakten ist es, die Gründe festzuhalten, die zu Entscheidungen in der Softwareentwicklung geführt haben (engl.: Rationale Management). Werden diese Informationen festgehalten, so können beispielsweise auch dann noch Entscheidungen nachvollzogen werden, wenn der entsprechende Mitarbeiter nicht mehr verfügbar ist.

Wichtig ist die Dokumentation von Traceability-Informationen insbesondere in geografisch verteilten Softwareentwicklungsprojekten, da hier eine reibungslose Kommunikation nicht immer möglich ist. Des Weiteren kann durch das Ausscheiden von Mitarbeitern Wissen über die Zusammenhänge der Arbeit verloren gehen (engl.: tacit knowledge). Traceability und Rationale tragen damit sowohl zur effizienteren Durchführung als auch zum besseren Verständnis verteilter Softwareprojekte bei.

Werden allerdings alle Traceability-Informationen festgehalten, so entsteht unter Umständen einen Überfluss an Informationen, der nur noch schwer zu durchschauen ist. Einige Anforderungen sind möglicherweise nicht so wichtig wie andere, werden diese aber gleichrangig neben den Wichtigeren dargestellt, so kann der Überblick verloren gehen. Daher sollte Traceability- und Rationale-Management nutzen- oder wertbezogen (engl.: value-based) betrieben werden. Unter dem Wert wird in diesem Zusammenhang der Nutzen verstanden, mit dem ein Nutzer der Software bzw. der Auftraggeber eine Anforderung bewertet. Die Bewertung einer Anforderung durch die Entwickler kann von der Bewertung durch die Anwender deutlich abweichen, daher sollte der Hauptfokus bei der Entwicklung auf dem Nutzen für den Kunden liegen. Durch die Verknüpfung der Anforderungen mit den zugehörigen Artefakten werden wichtigere Artefakte auf diese Weise automatisch höher priorisiert.

Nutzenbezogenes Traceability- und Rationale-Management führt so unter anderem zu einem besseren Verständnis des Codes, zu besserer Wartbarkeit und Dokumentation, zur Qualitätssicherung, Fehlerreduktion und höherer Effizienz der Entwicklung sowie letztendlich zu höherer Kundenzufriedenheit.

1.2 Zielsetzung

Ziel dieses Artikels ist es, die Entwicklung eines Werkzeug vorzustellen, das Daten über Anforderungen, Aktivitäten und Nutzer, die in einer kollaborativen Softwareentwicklungsplattform verwaltet werden, extrahiert und die Beziehungen dieser Entitäten untereinander visualisiert und analysiert. Dabei sollen entsprechend dem Wert der Anforderungen und daraus resultierender Artefakte wichtige Beziehungen besonders hervorgehoben werden. Zusätzlich sollen die Rationale-Informationen von verschiedenen Artefakttypen dargestellt und analysiert werden können.

1.3 Vorgehensweise und Gliederung

Als Kollaborationsplattform und Datenquelle wird dabei CodeBeamer von Intland[1] verwendet, während zur Bewertung der Anforderungen und Generierung der Nutzenwerte "*ibere*" verwendet werden soll, ein Werkzeug, das im Rahmen eines Forschungsprojekts an der Universität Mannheim entwickelt wurde (siehe dazu Geisser u. Hildenbrand (2006a) sowie Geisser u. Hildenbrand (2006b)). Der Name des hier implementierten Werkzeugs "TraVis" leitet sich aus **Tra**ce **Vis**ualization ab.

Zunächst werden die Konzepte des Value-Based Software Engineering (VBSE), des Traceability- und des Rationale Managements sowie deren Zusammenspiel theoretisch eingeführt.

Danach wird im praktischen Teil der Arbeit in Kapitel 3 auf die Anforderungserhebung sowie in Kapitel 4 auf Implementierungsdetails eingegangen. Dabei werden zunächst die Basistechnologien - die CodeBeamer Plattform und das JUNG-Framework[2] - vorgestellt. Im Anschluss werden Details der Architektur sowie die grundlegenden Algorithmen erläutert.

Anschließend folgen eine Installations- und Bedienungsanleitung von TraVis sowie die Vorstellung einer Beispielinstallation. Im letzten Kapitel folgt neben einer Zusammenfassung ein Ausblick auf Einsatzmöglichkeiten sowie künftige Verbesserungsmöglichkeiten von TraVis.

[1] http://www.intland.com (23.02.2007)
[2] Java Universal Network/Graph Framework: http://jung.sourceforge.net (23.02.2007)

2 Wertbasiertes Traceability- und Rationale-Management

In diesem Kapitel werden kurz die grundlegenden Konzepte sowie Vor- und Nachteile von Traceability, Rationale Management und des Value-Based Software Engineering vorgestellt.

2.1 Traceability

Unter Nachvollziehbarkeit versteht man die Möglichkeit zur Verfolgung einer Systemanforderung in vorwärts- und rückwärtsgerichteter Richtung. So sollten die Anforderungen von ihrem Ursprung über ihre Entwicklung in Designdokumenten und ihre Umsetzung im Code bis hin zur Verifikation nachvollziehbar sein (Gotel u. Finkelstein (1994)).

Man kann dabei zwischen Pre-Traceability (Quellennachvollziehbarkeit), Anforderungsnachvollziehbarkeit und Post-Traceability (Entwurfsnachvollziehbarkeit) unterscheiden. Pre-Traceability findet vor der eigentlichen Anforderungserstellung statt, es sind hier also die Beziehungen zwischen den Quellen der Anforderungen (Stakeholder, technische Annahmen, kooperierende Systeme, Gesetze usw.) und den Anforderungen gemeint. Vorwärtsgerichtet heisst, dass die Quelle mit der entsprechenden Anforderung verbunden wird, wohingegen rückwärtsgerichtet bedeutet, dass die Anforderungen mit ihren Quellen verbunden werden. Mit Anforderungsnachvollziehbarkeit bezeichnet man die Beziehungen zwischen den verschiedenen Anforderungen.
Die Beziehungen zwischen den Anfordeungen und den daraus resultierenden Artefakten, die erst nach der Anforderungserstellung erzeugt werden können, werden unter dem Begriff Post-Traceability zusammengefasst.
Vorwärtsgerichtet sind hier die Beziehungen von den Anforderungen zu den Artefakten; mit rückwärtsgerichtet bezeichnet man dagegen die Anforderungen zwischen Artefakten und Anforderungen (vgl. Virtuelles Software Engineering Kompetenzzentrum).

Lindvall u. Sandahl (1996) unterscheiden zusätzlich vertikale Traceability von horizontaler Traceability. Mit vertikaler Traceability meinen die Autoren hier die Nachvollziehbarkeitsbeziehungen innerhalb der Anforderungen, den Analyse- und Designdokumenten oder dem Code. Horizontale Traceability bezieht sich hingegen auf die Beziehungen zwischen diesen Kategorien, also beispielsweise zwischen Anforderungs- und Analysedokumenten.
Sommerville (2007) definiert die Nachvollziehbarkeit hingegen als "[...] the property of a requirements specificaion that reflects the ease of finding related requirements" (vgl. Sommerville (2007), Seite 163). Dabei unterscheidet er zwi-

schen “Source Traceability”, “Requirements Traceability” und “Design Traceability”. Source Traceability deckt sich dabei mit der Pre-Traceability. Requirements Traceability bezieht sich ähnlich wie die vertikale Traceability bei Lindvall und Sandahl auf Abhängigkeitsbeziehungen innerhalb einer Kategorie, insbesondere zwischen den Anforderungen. Design Traceability verknüpft schließlich die Anforderungsdokumente mit den daraus resultierenden Designdokumenten. Dies entspricht der Post-Traceability. Die Autoren visualisieren die Abhängigkeitsbeziehungen als Graph in einem Traceability Netzwerk.

Erforderlich wird Traceabilitymanagement (TM) dadurch, dass der Trend zu mehr verteilten Softwareentwicklungsprojekten geht (Heindl u. Biffl (2006)). Durch die verteilte Entwicklung steigt der Kommunikationsaufwand und der Bedarf nach Nachvollziehbarkeitsinformationen. Arbeiten viele Mitarbeiter an verschiedenen Orten und möglicherweise in verschiedenen Zeitzonen, so bekommt die asynchrone Kommunikation eine höhere Bedeutung (Dutoit u. a. (2006)).

Einige Fragen können nur über synchrone Kommunikation geklärt werden, diese ist in verteilten Projekten aber teurer als asynchrone Kommunikation und schränkt die Vorteile der verteilten Entwicklung ein (zum Beispiel Entwicklung rund um die Uhr oder Arbeit der Mitarbeiter von zu Hause aus). Um den Bedarf an synchroner Kommunikation zu verringern sollte eine Anforderungsverfolgung (engl.: Requirements tracing) betrieben werden, um die Anforderungen für alle verständlich und nachvollziehbar zu machen (vgl. dazu Heindl u. Biffl (2006)).

Die Nachvollziehbarkeit ist deshalb wichtig, damit die Auswirkungen von Änderungen klar sind. Wird in einer späten Projektphase eine Anforderung geändert, so sollte sofort für alle Beteiligten ersichtlich sein, welche Auswirkungen die Änderungen im Projekt bewirken, beispielsweise, welche Designdokumente angepasst werden müssen oder welche Softwarekomponenten betroffen sind (Heindl u. Biffl (2006)). Genauso wichtig ist es, einfach feststellen zu können, welche Mitarbeiter über Änderungen am Design informiert werden müssen, beispielsweise weil sie gerade an dem Code arbeiten, der von den Änderungen betroffen ist. Für den Softwareentwickler ist es wichtig, dass er jederzeit nachvollziehen kann, warum eine Entscheidung getroffen wurde, beziehungsweise welche Anforderungen hinter den Designdokumenten oder sonstigen Artefakten (beispielsweise Quellcode) stehen.

Weitere Gründe, die das TM notwendig machen, finden sich unter anderem bei Egyed (2006), Marcus u. a. (2005) und Antoniol u. a. (2002). Beispielsweise dient die Erfassung der Beziehungen der Artefakte der Dokumentation und damit dem besseren Programmverständnis. Dem Entwickler nützen die Informationen für eine einfachere Auswirkungsanalyse, für Konsistenzchecks sowie zur Unterstützung des Testens. Außerdem kann die Wiederverwendung der Artefakte leichter erreicht werden, so zum Beispiel wenn Anforderungen sich in verschiedenen Projekten ähneln. Weiterhin hilft TM die Kosten abzuschätzen und Fehler zu vermeiden oder zu reduzieren, da Anforderungen klarer sind. Letztendlich kann TM durch die gestiegene Qualität zu höherer Kundenzufriedenheit führen (Egyed (2006)).

Im Umkehrschluss ergeben sich viele negative Nebeneffekte, wenn man die Abhängigkeitsbeziehungen nicht versteht. In erster Linie steigert dies das Risiko, dass Änderungen nicht überall übernommen werden, wo es nötig ist. So ist einem Softwarearchitekten möglicherweise nicht klar, welche anderen Artefakte von seinen Änderungen betroffen sind und er vergisst eventuell, einige Artefakte anzupassen. Außerdem verursacht es Fehler dadurch, dass Entwickler Inkonsistenzen nicht wahrnehmen oder ignorieren und ihre Entscheidungen aufgrund ungenauer Informationen treffen (siehe auch Egyed (2006)).

Die Abhängigkeitsanalyse ist auch deshalb wichtig, weil Systeme und Software gleichzeitig entwickelt werden. Gleichzeitige Entwicklung beinhaltet, dass Änderungen jederzeit und überall auftreten können. Eine Rückverfolgbarkeit (engl.: Trace) zwischen den Artefakten hilft, die Auswirkungen der Änderungen über alle Entwicklungsartefakte (zum Beispiel Anforderungsdokumente, Design, Implementierung) zu identifizieren. Die Rückverfolgbarkeit ist auch für die wertbezogene Überwachung und Kontrolle wichtig (vgl. Boehm (2003)), weil der Entwickler die Abbildung von Ziel und Lösung verstehen muss. Dieser Wertgewinn wurde erkannt, so dass es viele Standards gibt, die Trace-Analyse als benötigte Aktivität vorschreiben (unter anderem ISO 9000-3 (1991), IEEE 1219 (1992), DOD-STD-2167A (1988) und CMMI (2002), vergleiche dazu auch Egyed (2006)). Insbesondere wird Traceability in diesen Standards als Qualitätsmerkmal gesehen.

Ergänzend seien hier noch zwei Definitionen erwähnt:
IEEE 610[1]:"The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another."
SEI CMMI[2]: "The evidence of an association between a requirement and its source requirement, its implementation, and its verification."

Das SWEBOK (2004) geht ebenfalls auf Traceability ein, insbesondere ist dort Traceability aus vertraglichen Gründen wichtig um nachzuweisen, dass alle Anforderungen umgesetzt wurden. Auch die Notwendigkeit von Traceability bei Anforderungsänderungen wird im SWEBOK hervorgehoben. Entwickler haben nur ein begrenztes Verständnis von der Software, die sie nicht selber entwickelt haben. Daher ist Traceability notwendig, um dieses Verständnis zu erweitern. Des Weiteren wird Traceability auch im SWEBOK als Qualitätsmerkmal gesehen.

Erschwert wird TM dadurch, dass anders als bei anderen Problemen des "Reverse Engineering" die Aufdeckung von Traceability Links zwischen freier Textdokumentation, UML-Diagrammen und Quellcode-Komponenten nicht einfach auf Compilertechniken basieren kann, da die Anwendung einer syntaktischen Analyse auf Sätze der natürlichen Sprache schwierig ist (vgl. Antoniol u. a. (2002)). Eine Automatisierung ist daher nur schwer umsetzbar.
Bei dem Ansatz, der dieser Arbeit zugrunde liegt, müssen die Abhängigkeitsbe-

[1] vgl. IEEE 610 (1990)
[2] vgl. CMMI (2002)

ziehungen manuell in der Kollaborationsplattform eingepflegt werden, um analysiert werden zu können.

2.2 Rationale Management

Unter Rationale- (engl.: Begründungs-) Management (im Folgenden RM) versteht man die Erfassung und Dokumentation der Gründe, die zu einer Entscheidung geführt haben. Die Begründung enthält dabei die Probleme, auf die die Entwickler gestoßen sind, die Lösungsmöglichkeiten, die gefunden wurden, die Kriterien, nach denen die Optionen bewertet wurden sowie die Diskussion, die zur Entscheidungsfindung geführt hat (vgl. Dutoit u. Paech (2001) sowie Dutoit u. a. (2006)).

Design Rationale (DR) ist eine Verfeinerung des RM, hierbei werden nur die Entscheidungen erfasst, die sich mit den Entscheidungen des Designs beschäftigen, nicht jedoch Entscheidungen, die beispielsweise die Umsetzung des Designs in konkreten Code betreffen. Lee (1997) definiert DR wie folgt: "Design rationales include not only the reasons behind a design decision but also the justification for it, the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision" (vgl. Lee (1997), Seite 78)

Die Begründung kann dabei zwei verschiedenen Zwecken dienen, der Diskussion und der Wissenserfassung:
Indem es die Gründe offen legt, die zu Entscheidungen führen, erleichtert RM die Diskussionen zwischen Entwicklern, da die Alternativen und deren Bewertung systematisch erfasst werden. Außerdem explizieren die Entwickler durch das Erfassen ihrer Gründe Wissen, das normalerweise nur implizit vorhanden ist. Später kann dann die Ursache für bestimmte Entscheidungen überprüft werden, zum Beispiel, wenn das System aufgrund der Verfügbarkeit von neuen Technologien geändert werden soll (Dutoit u. Paech (2001))

Im Folgenden sollen die Vorteile eines effektiven RM dargestellt werden, die ähnlich zu denen für das Tracebility-Management sind (vergleiche dazu Lee (1997), Dutoit u. Paech (2001), Dutoit u. a. (2006) sowie Horner u. Atwood (2006)):
Ein Vorteil besteht in der besseren Zusammenarbeit: Durch das Festhalten aller Alternativen, Bewertungen und Gründe, die zu Entscheidungen geführt haben, kann jeder Projektmitarbeiter jederzeit Entscheidungen nachvollziehen und zudem hat jeder Mitarbeiter den gleichen Wissensstand, was letztlich zu einer besseren Zusammenarbeit führt. Dadurch wird auch das Projektmanagement verbessert, da der Bedarf an synchroner Kommunikation sinkt. Insbesondere in verteilten Entwicklungsprojekten ist die asynchrone Kommunikation wichtig. Sitzt das Projektteam im selben Raum, so ist es kein Problem, sofort über Unklarheiten zu sprechen. Sind die Projektmitarbeiter jedoch global in verschiedenen Zeitzonen verteilt, so kann eine Unklarheit sofort zu Verzögerungen führen. Wenn jedoch alle relevanten Informationen zentral erfasst sind, so besteht diese Gefahr nicht.
Außerdem führt RM zu besseren Entscheidungen: Da im Idealfall alle möglichen

Alternativen erfasst, bewertet sowie gegeneinander abgewogen werden, steigt die Qualität dieser Entscheidungen. Mit den besseren Entscheidungen geht auch eine verbesserte Qualität einher: An der Softwareentwicklung sind immer verschiedene Stakeholder mit unterschiedlichem Hintergrund und unterschiedlichen Zielen beteiligt. Durch die oben genannte verbesserte Zusammenarbeit sowie den verbesserten Entscheidungen steigt insgesamt die Qualität der Software.
Auch das Abhängigkeitsmanagement wird durch RM verbessert: Da die Entscheidungen besser dokumentiert sind, steigt die Qualität des Abhängigkeitsmanagements (siehe Kapitel 2.1).
Ferner hat TM Einfluss auf die Dokumentation eines Projektes: Durch RM ist das gesamte Projekt besser dokumentiert, dadurch ist es auch einfacher, Dokumentationen zu erstellen und es fällt neuen Projektmitarbeitern leichter, sich Wissen über das Projekt anzueignen, da sie das Projekt einfacher nachvollziehen können. Zusätzlich ist es durch die verbesserte Dokumentation einfacher, Komponenten wiederzuverwenden.
Damit geht die verbesserte Nachvollziehbarkeit einher: Erfasst man nicht nur die Entscheidungen, sondern auch deren Abhängigkeiten und die Begründungen, so erleichtert das die Reevaluation in späteren Entwicklungsphasen.
Softwaresysteme haben einen langen Lebenszyklus oder sind in Systeme mit einem langen Lebenszyklus eingebettet. Folglich müssen Entwickler und das Wartungspersonal oft mit Entscheidungen arbeiten, die in der Vergangenheit von Teilnehmern getroffen wurden, die nicht mehr Teil des Projektes sind. Sind die Entscheidungen gut dokumentiert, so wird die Wartung wesentlich erleichtert.
Als weiterer Vorteil sind außerdem Kostenvorteile zu nennen: RM verursacht zwar in frühen Entwicklungsphasen einen hohen Arbeitsaufwand und dadurch hohe Kosten, in späteren, teureren Phasen (z.B. der Wartungsphase) sinken jedoch die Kosten erheblich.

Nach Heindl u. Biffl (2006) gibt es zwei Wege, um die Beziehungen zwischen Anforderungen und Entwicklungsartefakten in großen und komplexen Projekten zu erfassen:
(a) Indem die Begründungen in Dokumenten erfasst werden. Diese kommen normalerweise aus der synchronen Kommunikation, z.B. von einem Workshop, oder werden von einem zentralen Schriftführer erfasst.
(b) Durch Zugriff auf weniger strukturierte Dokumente, die aus der asynchronen Projektkommunikation entstehen, wie beispielsweise E-Mails oder informelle Memos.

Dutoit u. a. (2006) unterscheidet beim ersten Punkt noch zwischen der Erfassung durch die Beteiligten selbst und der Erfassung durch Spezialisten.

RM hat jedoch auch einige nicht zu vernachlässigende Nachteile. So verursacht es, wie bereits erwähnt, in frühen Entwicklungsphasen höhere Kosten (vgl. Dutoit u. Paech (2001)), verlängert aber zum Beispiel auch die Projektdauer. Diese Faktoren müssen gegen die späteren Vorteile abgewogen werden. Nicht jede Art von Projekten profitiert von einer solchen Investition. In kleinen, wenig komplexen Projekten kann es effektiver sein, auf ein RM zu verzichten, um beispielsweise eine schnellere Zeit bis zur Marktreife zu erreichen. Die Zeit, die in das

RM gesteckt wird bringt gerade in kleineren Projekten möglicherweise weniger Nutzen, als wenn sie direkt in das Design oder in die Implementierung gesteckt werden würde (vgl. Dutoit u. a. (2006)).
Dadurch kann auch die Motivation für solchen Mehraufwand gering sein, was daraufhin einen negativen Einfluss auf die Qualität der Informationen haben kann. Die Motivation der Mitarbeiter sinkt möglicherweise auch dadurch, dass die Nutzer der Informationen normalerweise andere Personen sind (beispielsweise das Wartungspersonal). Wenn die Mitarbeiter keinen eigenen Vorteil im RM sehen, so ist es wahrscheinlich, dass sie keinen großen Aufwand betreiben, um ihre Entscheidungsfindung zu dokumentieren (vgl. Dutoit u. Paech (2001)). Außerdem wird der natürliche Ablauf der Designaktivitäten unterbrochen, da neue Arbeitsschritte hinzukommen. Zusätzlich besteht die Gefahr, dass man durch zu viele zusätzliche Informationen den Überblick verliert (vgl. Lee (1997)).
Ein anderer Aspekt ist, dass Entwickler auch eine gewisse Ablehnung gegen das RM entwickeln könnten, beispielsweise aus politischen oder rechtlichen Gründen. So kann es sein, dass ein Entwickler nicht will, dass sein Vorgesetzter oder andere Personen seine Gründe für die getroffenen Entscheidungen kennt. Es ist außerdem denkbar, dass Entwickler sich gegen mögliche rechtliche Verfahren absichern wollen. Die Offenlegung aller Gründe und auch der Alternativen liefert anderen Mitarbeitern zudem die Möglichkeit, die getroffenen Entscheidungen zu kritisieren (vgl. Dutoit u. a. (2006) sowie Horner u. Atwood (2006)).
Ein weiteres Problem ist, dass die Rationale-Informationen viele Artefakte und Abhängigkeitsbeziehungen betreffen, was es schwierig macht, sie stets auf dem aktuellen Stand zu halten. Rationale-Informationen gehen verloren, wenn sie nicht zeitnah zu dem Problem, das sie betreffen, erfasst werden. Daher ist es schwierig, die Vollständigkeit zu gewährleisten.
Durch das RM steigt auch die Komplexität, da eine größere Menge an Informationen vorhanden ist. Wenn der Zugriff auf diese Informationen zu kompliziert wird, dann wird das System kaum genutzt werden (vgl. Dutoit u. Paech (2001)).

2.3 Value-Based Software Engineering

Heutzutage wird die meiste Arbeit im Software-Engineering noch wertneutral betrieben, das heißt, dass jede Anforderung, jeder Anwendungsfall, jedes Objekt, jeder Testfall und jeder Fehler als gleich wichtig betrachtet wird (Boehm (2006)).

Beim Value-Based Software Engineering (VBSE) geht man jedoch davon aus, dass nicht jedes Softwareartefakt den gleichen Wert hat (siehe auch Egyed u. a. (2005)). Unter dem Wert versteht man hier den Nutzen, den ein Artefakt (in erster Linie Anforderungen an die Software) für einen Stakeholder hat. Ein Softwareentwicklungsprojekt kann dabei viele Arten von Stakeholdern haben, unter anderem Kunden, Vorstände, Projektleiter, Entwickler, Marketingmitarbeiter, Verkäufer usw. Bei VBSE sind die wichtigsten Stakeholder die Kunden. Deren Bewertung des Wertes der Anforderungen ist für die Entwicklung ausschlaggebend.

Einige Studien, beispielsweise der CHAOS Report der Standish Group (Standish Group (1994)) haben herausgefunden, dass Softwareprojekte unter anderem daran scheitern, dass kaum wertorientiert gehandelt wird. Weitere Gründe für das Scheitern von Softwareprojekten sind zum Beispiel mangelnder User Input, unvollständige Anforderungen, fehlende Ressourcen, unrealistische Erwartungen und Zeitfenster sowie unklare Ziele.

Eine Ursache dafür zeigt das folgende Beispiel: Angenommen, es gibt fünf Anforderungen, A-E. Die Implementierung jeder dieser Anforderungen dauert eine Woche, insgesamt hat man drei Wochen Zeit. Die Anforderungen A und B werden vom Kunden als besonders wichtig angesehen, D und E sind nicht zwingend erforderlich, aber wesentlich einfacher zu implementieren. Wird nun nicht wertorientiert vorgegangen, so würde der Entwickler möglicherweise zunächst die einfacheren Anforderungen D und E umsetzen, am Ende bliebe zu wenig Zeit für die wichtigen Anforderungen A und B. Bei VBSE umgeht man dieses Problem, da der Kunde A und B mit einer höheren Wichtigkeit bewertet hat. Somit werden diese zuerst implementiert und die Bestandteile, die zeitgerecht fertig werden, liefern dann den höchsten Nutzen für den Kunden.

Eine weitere Ursache könnte sein, dass sich Projektmanagementsysteme zumeist nur auf die Projektkosten und den Zeitplan konzentrieren aber nicht auf den Wert für die Stakeholder oder das Unternehmen eingehen. Zudem wird die Verantwortung der Softwareentwickler oft lediglich in der Umsetzung der Anforderungen in Code gesehen. Softwareentscheidungen hatten vor einiger Zeit nur relativ geringen Einfluss auf die Systemkosten, den Zeitplan und den Wert des Systems, so dass die wertneutrale Entwicklung funktioniert hat. Dies ist aber heute nicht mehr zeitgemäß, da Software einen steigenden Einfluss auf diese Dinge hat, so dass ein Ansatz, der den Wert der einzelnen Anforderungen einbezieht immer wichtiger wird (Boehm (2006)).

Die Notwendigkeit von VBSE wird auch dadurch deutlich, dass laut Boehm (2006) 80% des Wertes einer Software durch 20% der Softwarekomponenten verursacht werden. Es sollte daher das Ziel sein, diese 20% der Komponenten in jedem Fall umzusetzen.

Nach Berry u. Aurum (2006) kann man den Wert nur dann managen, wenn man ihn auch messen kann. Für VBSE trifft das nur eingeschränkt zu, da man den Wertgewinn, den ein Stakeholder einer Anforderung zuschreibt, nicht unbedingt quantifizieren kann. Dies ist in erster Linie nur bei nichtfunktionalen Anforderungen (beispielsweise Performance) möglich, nicht aber bei funktionalen.

Durch die Wertüberlegungen von VBSE können Entscheidungen auf allen Ebenen der Softwareentwicklung so optimiert werden, dass sie die Ziele der involvierten Stakeholder treffen. Dabei werden Entscheidungen nicht blind nach individuellen Präferenzen getroffen, sondern stets so, dass sie den Wert maximieren (Biffl u. a. (2006)).

Im Kontext der Nachvollziehbarkeit geht man dementsprechend davon aus, dass nicht alle Abhängigkeitsbeziehungen gleich wichtig sind. Dadurch sollten sich

die Kosten reduzieren lassen, da der Fokus auf den Artefakten liegt, die den höchsten Stakeholdervalue liefern.
VBSE fügt verschiedenen Softwareartefakten Wert hinzu. Zum Beispiel könnten Anforderungen als kritisch, wichtig oder wünschenswert klassifiziert werden. Selbst wenn die wünschenswerten Anforderungen implementiert werden, ist deren Korrektheit nicht so wichtig wie die der kritischen Anforderungen (vgl. dazu Egyed u. a. (2005)).

Wird das VBSE mit der in Kapitel 2.1 vorgestellten Nachvollziehbarkeit kombiniert, so ist es auch in späteren Projektphasen einfach, die wirklich wichtigen Artefakte zu identifizieren und zu entwickeln. Insbesondere ist dies dadurch möglich, dass der Wert der Anforderungen über nachfolgende Artefakte bis hin zu den Artefakten, die am Ende tatsächlich entwickelt werden sollen, übertragen wird (Transitivität). Durch diese Kombination der beiden Konzepte können schließlich bessere projektinterne Entscheidungen getroffen werden (vgl. Heindl u. Biffl (2006)).

3 Anforderungserhebung für TraVis

In diesem Kapitel sollen die Anforderungen an das zu entwickelnde Werkzeug erläutert sowie eine Abgrenzung zu anderen Arbeiten in Bezug auf wertbasierte Nachvollziehbarkeits- und Rationale-Management vorgenommen werden.

3.1 Anforderungen

Wie bereits in der Einleitung zu dieser Arbeit erläutert, soll es sich bei dem zu entwickelnden Programm um ein Werkzeug handeln, dass die Möglichkeit bietet, Daten, die in einer Kollaborationsplattform (vgl. dazu Abschnitt 4.1.1)gespeichert sind, auszulesen und nach Aspekten des wertbasierten Nachvollziehbarkeitsmanagements darzustellen. Zusätzlich sollen die Rationale-Informationen, die über einzelne Artefakte verfügbar sind, übersichtlich veranschaulicht werden.

Nach einem ersten Brainstorming in einem Wiki-System wurden alle Anforderungen mit dem an der Universität Mannheim entwickelten Werkzeug "*ibere*" bewertet (Geisser u. Hildenbrand (2006a,b)). Hierdurch konnten weniger wichtige Anforderungen verworfen werden. Die restlichen Anforderungen wurden ausgewählt und werden im Folgenden präsentiert.

Das Werkzeug soll dem Nutzer die Möglichkeit geben, Daten aus der Kollaborationsplattform auszulesen und zu verwerten. Dazu soll er sich mit seinen Zugangsdaten in die Plattform einloggen können und anschließend die Möglichkeit haben, ein Projekt aus den Projekten, bei denen er angemeldet ist, auszuwählen.

Die verschiedenen Artefakttypen (beispielsweise Nutzer, Aufgaben, Dokumente, Wiki-Seiten oder Quellcode) sollen in einem Graphen dargestellt werden, wobei jedes Artefakt einem Knoten im Graphen entspricht. Beziehungen zwischen den Artefakten sollen durch Kanten zwischen den Knoten dargestellt werden.
Der Nutzer soll die Möglichkeit haben, Kategorien von Knoten und Kanten ein- oder auszublenden. Zusätzlich soll es die Möglichkeit geben, alle Knoten anzuzeigen, die mit einem Knoten verbunden sind.

TraVis soll verschiedene Anzeigemodi umsetzen. Zum einen einen Modus, bei dem alle Knoten in der gleichen Größe angezeigt werden, zum anderen einen Modus, bei dem die Knoten verschiedene Größen haben. Die Größe der Knoten soll dabei aus den Werten berechnet werden, die den Anforderungen, die in der Plattform gespeichert sind, zugeordnet sind. Dadurch sollen die Ideen des VBSE umgesetzt werden. Die Knoten, die die Anforderungen repräsentieren, sollen ihre Größe entsprechend dem ihnen zugeordneten Wert ändern. Die Knoten, die mit den Anforderungen assoziiert sind, sollen eine Größe bekommen, die der Summe der mit ihnen verbunden Anforderungen entspricht. Dies gilt jedoch

nur für Dokumente, Quellcode und Aufgaben (engl.: TrackerItems). Knoten, die keinen Wert im eigentlichen Sinne des VBSE haben, wie beispielsweise Forenbeiträge, Nutzer oder Chatnachrichten, sollen ihre Größe nicht ändern.
In der Plattform gespeicherte TrackerItems, die keiner Anforderung zugeordnet sind, sollen eine Größe bekommen, die ihrer Priorität entspricht.

Zusätzlich soll TraVis alle Informationen, die für ein effektives Rationale-Management erforderlich sind, übersichtlich darstellen. Dazu soll es die Möglichkeit geben, einen Knoten auszuwählen, dessen Informationen daraufhin separat aufbereitet werden. So sollen beispielsweise bei einem TrackerItem die zugeordneten Diskussionsforen inklusive Forenbeiträgen, Wiki-Seiten, Dokumente oder Kommentare dargestellt werden.
Diese Möglichkeit soll es ebenfalls für Dokumente und Quellcode geben.

3.2 Bezug zu anderen Arbeiten

In Kapitel 2.1 wurde bereits erwähnt, dass die Abhängigkeitsbeziehungen bei dem Ansatz, der dieser Arbeit zu Grund liegt, manuell in die Plattform eingepflegt werden müssen, damit eine Analyse möglich ist.

Es gibt jedoch auch Ansätze, die Abhängigkeitsbeziehungen automatisch auslesen. So haben beispielsweise Egyed u. a. (2005) mit Trace/Analyzer ein Werkzeug entwickelt, das automatisch Abhängigkeitsbeziehungen herstellt, wenn sich der Code verschiedener Artefakte überschneidet. Allerdings können hier lediglich Beziehungen zwischen dem Code verschiedener Artefakte dargestellt werden; eine Nachvollziehbarkeit bis zu den Anforderungsdokumenten, wie es für die vollständige Umsetzung des Nachvollziehbarkeitsmanagements notwendig ist, ist hier nicht möglich.

Marcus u. a. (2005) haben mit TraceViz ebenfalls ein Werkzeug zur Darstellung von Traceability-Informationen entwickelt. Dieses wird als Plugin in die Entwicklungsplattform Eclipse[1] integriert. TraceViz nutzt eine Link-Semantik, um Abhängigkeitsbeziehungen zwischen dem Code eines Projektes und seiner externen Dokumentation herstellen.

Ariadne (vgl. de Souza u. a. (2004) und Trainer u. a. (2005)) ist ebenso wie TraceViz eine Erweiterung der Entwicklungsumgebung Eclipse. Mit Hilfe von Ariadne können Abhängigkeiten im Quellcode von Java-Programmen (Technische Abhängigkeiten) aufgedeckt werden und als sogenannte „call-graphs“ visualisiert werden. Zusätzlich können die sozialen Netzwerke (Soziale Abhängigkeiten, so genannte „social network graphs“), die sich aus den miteinander verknüpften Klassen ergeben, dargestellt werden. Um diese Netzwerke aufzudecken, wird das Repository im Dokumentenmanagementsystem, mit dem das Projekt verwaltet wird, analysiert. Dabei werde zunächst die Beziehungen zwischen dem Code und einzelnen Personen (Sozio-Technische Abhängigkeiten) als „social call-graphs“ aufbereitet.

[1] `http://www.eclipse.org` (5.10.2006)

Lindvall u. Sandahl (1996) haben die Anwendung eines Traceability-Ansatzes in der Industrie untersucht. Dabei kam Objectory SE (Objective Systems SF AB (1993)) zum Einsatz, welches ebenso wie TraVis Traceability Beziehungen durch einen Graphen visualisiert.

Die Firma VA Software[2] hat schließlich mit dem SourceForge Explorer eine Software entwickelt, mit der sich Abhängigkeitsbeziehungen und Rationale-Informationen analysieren lassen. Die Beziehungen werden hier allerdings nicht durch einen Graphen visualisiert, im Vordergrund steht vielmehr die Möglichkeit, alle Funktionen der Kollaborationsplattform über eine Standalone-Anwendung zu bedienen, also beispielsweise neue Tracker anzulegen oder zu editieren[3].

Der geforderte Funktionsumfang für TraVis sowie die Umsetzung der Anforderungen in den anderen Werkzeugen sind zusammenfassend in Tabelle 3.1 dargestellt. Zusätzlich zu den vorgestellten Konzepten werden noch allgemeine Requirements-Engineering (RE-) Werkzeuge in der Übersicht aufgeführt, da solche Werkzeuge auch teilweise die Anforderungen an TraVis umsetzen.[4]

Anforderung	**TA**	**TV**	**AR**	**RE**	**TraVis**
Darstellung der Beziehungen aller Artefakte		X		X	X
Darstellung der Beziehungen des Codes untereinander	X	X	X		X
Soziale Netzwerke			X		X
Wertbasierte Darstellung					X
Verknüpfung mit Rationale-Informationen				X	X

Tabelle 3.1: Anforderungen an TraVis und Vergleich mit anderen Werkzeugen

[2] http://www.vasoftware.com (5.10.2006)

[3] Zum Funktionsumfang des SourceForge Explorers vergleiche VA Software (2006), Seiten 339 bis 375

[4] Legende: TA = Trace/Analyzer; TV = TraceViz; AR = Ariadne; RE = Requirements-Engineering Werkzeuge

4 Konzeption und Implementierung

In diesem Kapitel werden zunächst die verwendeten Technologien - die Kollaborationsplattform CodeBeamer und das JUNG-Framework - vorgestellt. Im Anschluss daran wird das Design von TraVis erläutert, wobei dabei insbesondere auf den Aufbau von TraVis, die verwendeten Datenstrukturen sowie die Methoden, die die Konzepte der Value-Based Traceability und des Rationale Managements umsetzen, eingegangen wird.

4.1 Basistechnologien

4.1.1 CodeBeamer

CodeBeamer ist eine Kollaborationsplattform, die von der Firma Intland[1] entwickelt wird.

Eine Kollaborationsplattform ist eine Plattform, die die Zusammenarbeit im Softwareentwicklungsprozess unterstützt. Dabei werden sowohl Kommunikationswerkzeuge als auch Projektmanagementfunktionen sowie Dokumentenmanagementsysteme in einer Plattform vereinigt. Funktionen, die eine solche Plattform liefern sollte, sind unter anderem Möglichkeiten zur Projektkoordination, also beispielsweise die Zuweisung von Aufgaben an Projektmitglieder, Erstellung von Zeitplänen oder Gantt Charts etc. Die Plattform sollte außerdem Möglichkeiten zur Versionskontrolle liefern, das Einbinden von Dokumenten ermöglichen, eine Problemverfolgung enthalten sowie die Erstellung von Berichten vereinfachen. Weitere Anforderungen sind die Bereitstellung von Kommunikationsmöglichkeiten, so beispielsweise Mailinglisten, Diskussionsforen, Wikis oder Möglichkeiten zur Echtzeitkommunikation, zum Beispiel die Einbindung eines Chats. Dabei sollen möglichst alle Informationen gespeichert werden, damit in späteren Projektphasen zur besseren Nachvollziehbarkeit wieder darauf zurückgegriffen werden kann.

Es gibt inzwischen eine Reihe von Kollaborationsplattformen auf dem Markt. Einige sind proprietär (d.h. geschützt und nicht frei einsehbar), beispielsweise SourceCast Enterprise von CollabNet[2] (CN), SourceForge Enterprise von VA Software[3] (SFE), Application Lifecycle von Borland[4] (AL) oder CodeBeamer von Intland (CB). Andere sind wiederum frei verfügbar, beispielsweise sour-

[1] http://www.intland.com (5.10.2006)
[2] http://www.collab.net (5.10.2006)
[3] http://www.vasoftware.com/sourceforge (5.10.2006)
[4] http://www.borland.com/de/products/alm/index.html (5.10.2006)

ceforge.net[5] (SFnet), GForge[6] (GF) oder Savane[7] (S), um hier nur einige zu nennen.
Es gibt außerdem die Möglichkeit, Entwicklungsplattformen, beispielsweise das bereits erwähnte Eclipse (E), durch Plugins zu erweitern.

In einer Marktstudie an der Universität Mannheim (vgl. Geisser u. a. (2006)) wurden verschiedene Kollaborationsplattformen miteinander verglichen. Dabei hat sich herausgestellt, dass CodeBeamer den größten Funktionsumfang der verglichenen Plattformen liefert. Hervorgehoben hat sich CodeBeamer insbesondere auch durch die RemoteApi, durch die der Zugriff auf die gespeicherten Daten relativ einfach möglich ist. Daten können über diese Schnittstelle nicht nur ausgelesen, sondern auch editiert und wieder in die Plattform geladen werden. Ein kurzer Vergleich des Funktionsumfangs der oben erwähnten Plattformen ist in Tabelle 4.1 dargestellt. Ein ausführlicherer Vergleich dieser, sowie einiger weiterer Plattformen, findet sich bei Geisser u. a. (2006).

	CN	**SFE**	**AL**	**CB**	**SF.net**	**GF**	**S**	**E**
Wiki		X		X				
Forum	X	X	X	X	X	X		
Tracker	X	X	X	X	X	X	X	
API		X		X				
Chat	X		X	X				
Versionskontrolle	X	X	X	X	X	X	X	X

Tabelle 4.1: Vergleich verschiedener Kollaborationsplattformen

Konkret liefert CodeBeamer unter anderem ein Wiki, die Möglichkeit verschiedene Tracker, das heisst Aufgabenlisten, anzulegen, ein Forum, einen Chat, Mailinglisten, eine Nachrichtenfunktion, ein integriertes Dokumentenmanagementsystem, eine Suchfunktion sowie eine E-Mail-Integration, durch die beispielsweise auch eine Benachrichtigungsfunktion ermöglicht wird. Außerdem besteht die Möglichkeit, externe Versionskontrollsysteme einzubinden, unter anderem die Open Source Systeme CVS[8] und Subversion[9] oder das von IBM entwickelte Rational ClearCase[10].

Durch die Möglichkeit, Assoziationen zwischen allen in der Plattform gespeicherten Artefakten zu erzeugen, liefert CodeBeamer auch wichtige Voraussetzungen für die Analyse der Traceability-Beziehungen. Unter Artefakten versteht man hier Dokumente, Aufgaben, Quellcode, Diskussionen in Foren usw. Auch die Voraussetzungen für ein effektives Rationale-Management sind durch diese Assoziationen erfüllt, da es dadurch möglich ist, beispielsweise Diskussionen, die zu einer Entscheidung geführt haben, einer Aufgabe zuzuordnen. Das Hinzufügen der Assoziationen kann dabei auf zwei verschiedene Arten erfolgen:

[5] http://www.sourceforge.net (5.10.2006)
[6] http://www.gforge.org (5.10.2006)
[7] https://gna.org/projects/savane (5.10.2006)
[8] http://www.nongnu.org/cvs (5.10.2006)
[9] http://subversion.tigris.org (5.10.2006)
[10] http://www.ibm.com/software/awdtools/clearcase (5.10.2006)

Zum einen durch das direkte Hinzufügen einer Assoziation bei dem entsprechenden Artefakt, zum anderen durch das Hinzufügen eines Links mittels der Wiki-Syntax.
Bei der ersten Methode kann man der Assoziation zusätzlich eine Semantik geben, so kann man den Typ der Assoziation mit einem der Attribute "depends", "parent", "child" oder "related" versehen. Diese Methode wird durch eine Suchfunktion unterstützt.

Die dieser Arbeit zu Grunde liegende Version von CodeBeamer ist Version 4.1.1.

4.1.2 JUNG

JUNG ist eine Open Source Java-Bibliothek, mit der Graphen und Netzwerke dargestellt werden können. Es wird an der University of California in Irvine[11] entwickelt. Die Abkürzung JUNG steht für Java Universal Network/Graph Framework.

Mit dem JUNG Framework können Daten, die in einem Graphen oder in einem Netzwerk dargestellt werden, modelliert, analysiert und visualisiert werden. Es unterstützt verschiedene Darstellungsmöglichkeiten, so zum Beispiel gerichtete und ungerichtete Graphen, Multimodale Graphen, Graphen mit parallelen Kanten und Hypergraphen. JUNG ist allgemein gehalten, so dass es für spezifische Aufgaben anpassbar und erweiterbar ist.

In JUNG sind verschiedene Algorithmen der Graphentheorie, des Data-Mining und der Social Network Analyse (beispielsweise Clusteranalyse, statistische Analyse) bereits enthalten. Zusätzlich liefert JUNG bereits verschiedene Layouts, mit denen die Graphen oder Netzwerke dargestellt werden können.

Neben JUNG gibt es noch eine Reihe weiterer Bibliotheken, die die Darstellung von Graphen unterstützen. Dabei konzentrieren sich einige auf die Darstellung von Graphen, so zum Beispiel Graphviz[12] (GV) oder Processing[13] (PC), andere sind eher für die allgemeinere Informationsvisualisierung geignet, so beispielsweise Prefuse[14] (PU), Piccolo[15] (PIC), The Visualization Toolkit (VTK)[16], oder The InfoVis Toolkit[17] (IV). Eine Übersicht zum Funktionsumfang dieser Bibliotheken findet sich in Tabelle 4.2. Andere Bibliotheken kamen aus verschiedenen Gründen nicht in die engere Auswahl, so beispielsweise Ucinet[18], da es nur proprietär ist, oder Improvise[19], welches nicht genug auf die Erzeugung von Graphen fokussiert.

[11] http://www.uci.edu
[12] http://www.graphviz.org/ (5.10.2006)
[13] http://processing.org (5.10.2006)
[14] http://www.prefuse.org (5.10.2006)
[15] http://www.cs.umd.edu/hcil/piccolo (5.10.2006)
[16] http://public.kitware.com/VTK (5.10.2006)
[17] http://ivtk.sourceforge.net (5.10.2006)
[18] http://www.analytictech.com/ucinet/ucinet.htm (5.10.2006)
[19] http://www.personal.psu.edu/faculty/c/e/cew15/improvise (5.10.2006)

	JUNG	PC	GV	PIC	PU	VTK	IV
Open Source	X	X	X	X	X	X	X
Aktive Entwicklergemeinschaft	X	X			X	X	
Vordefinierte Layouts	X		X	X	X		
Unterstützung von Graphen	X	X	X	X	X	X	X
Unterstützung von Netzwerken	X		X		X		
Unterstützung von Bäumen					X		X
Bereits vorhandene Algorithmen	X		X				
Zusatzfunktionen	X			X	X		
Gut dokumentiert	X	X	X	X	X	X	X
Einfachheit der Benutzung bzw. Einarbeitungsaufwand	X	X					X

Tabelle 4.2: Vergleich verschiedener Bibliotheken für die Visualisierung

JUNG wurde für diese Arbeit ausgewählt, da es als Open Source Bibliothek frei verfügbar und erweiterbar ist. Außerdem liefert es bereits einen sehr großen Funktionsumfang, der von den anderen Bibliotheken in diesem Maße nicht abgedeckt wird. Nützlich sind insbesondere bereits implementierte Zusatzfunktionen wie beispielsweise eine Zoomfunktion oder verschiedene Mausmodi. Außerdem hat JUNG eine große aktive Community, so dass damit zu rechnen ist, dass der Funktionsumfang noch steigen wird, was insbesondere für spätere Weiterentwicklungen des in dieser Arbeit entwickelten Werkzeugs ist. Ein weiterer positiver Aspekt an JUNG ist die Güte der Dokumentation, so gibt es zum einen die vollständige JavaDoc-Dokumentation, zum anderen aber auch Anwendungsbeispiele, die inklusive Quelltext zur Verfügung stehen. Dadurch sinkt der Einarbeitungsaufwand und die Benutzung wird deutlich vereinfacht. In dem in Abschnitt 3.2 vorgestellten Ariadne-Projekt wird JUNG ebenfalls eingesetzt.

Für TraVis ist die Verwendung einer solchen Bibliothek wichtig, da die Artefakte, die in CodeBeamer gespeichert sind, als Knoten in einem Graphen dargestellt werden sollen. Die Assoziationen, die jeder Knoten hat, sollen als Kanten zwischen den Knoten dargestellt werden.

4.2 Architektur

TraVis besteht aus insgesamt 5 Paketen. Eine Übersicht über die Pakete zeigt Abbildung 4.1. Im Paket `de.unimannheim.wifo1.travis.graphics` befinden sich die Graphiken, die in TraVis verwendet werden. Das Paket `de.unimannheim.wifo1.travis.utils` (siehe Abbildung 7.1, Seite 33) enthält Hilfsklassen, die zur Unterstützung der Visualisierung benötigt werden. Die Pakete

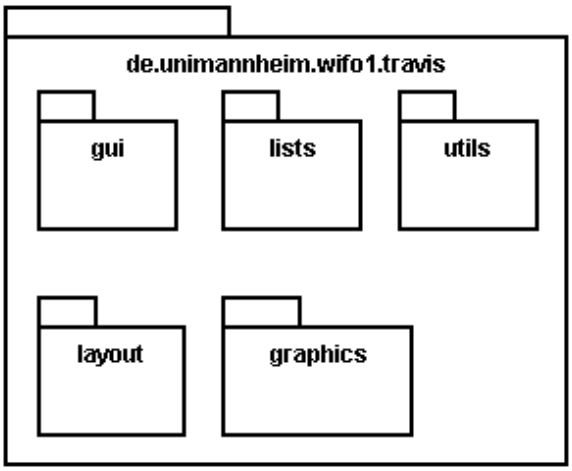

Abbildung 4.1: Paketdiagramm

`de.unimannheim.wifo1.travis.lists` (siehe Abbildung 7.2, Seite 34) und `de.unimannheim.wifo1.travis.layout` (siehe Abbildung 7.3, Seite 35) enthalten die Datenstrukturen (siehe Abschnitt 4.3.1) beziehungsweise die Algorithmen für das Layout des Graphen, mit dem die Abhängigkeitsbeziehungen visualisiert werden. Schließlich enthält das Paket `de.unimannheim.wifo1.travis.gui` (siehe Abbildung 7.4, Seite 36) die Klassen, die die Anwendungsoberfläche implementieren. Die Hauptklasse, mit der das Programm gestartet wird, befindet sich im Paket `de.unimannheim.wifo1.travis`. Eine Übersicht über die Klassen sowie eine kurze Beschreibung ihrer Funktionalität ist in Tabelle 4.3 aufgeführt. Eine ausführliche Dokumentation befindet sich außerdem als Java Doc im Ordner "Travis/doc" auf der beiligenden CD.

Zur besseren Lesbarkeit des Programmcodes wurden einige Regeln festgelegt: So beginnen Klassennamen mit einem Großbuchstaben, Variablen und Methodennamen dagegen mit einem Kleinbuchstaben. Klassen, Variablen und Methoden sind zudem intuitiv benannt, damit ihre Funktion einfacher festgestellt werden kann. Zusätzlich soll die Verwendung von CamelCase[20] die Lesbarkeit erleichtern.

Paket `de.unimannheim.wifo1.travis`	
Klasse	**Beschreibung**
TraVisMain	Startet TraVis

Paket `de.unimannheim.wifo1.travis.gui`	
TraVisLoginScreen	Anmeldefenster
TraVisMainScreen	Hauptfenster: Enthält die Grafik und die Logik
RationaleInformationScreen	Fenster, dass die Rationale-Informationen anzeigt
InfoScreen	Zeigt Versionsinformationen an

Paket `de.unimannheim.wifo1.travis.layout`

[20] CamelCase bedeutet, das jedes neue Wort in einem zusammengesetzten Wort mit einem Großbuchstaben beginnt. Beispiel: AutoWerkstatt

Klasse	**Beschreibung**
Klasse	**Beschreibung**
AbstractTraVisLayout	Abstrakte Layoutklasse
CircleLayout	Implementiert das Kreislayout für den Graphen
ISOMLayout	Implementiert das isometrische Layout für den Graphen
ChaosLayout	Implementiert eine ungeordnete Darstellung des Graphen

Paket `de.unimannheim.wifo1.travis.lists`	
Klasse	**Beschreibung**
AbstractList	Eine abstrakte Liste, von der die anderen Listen erben
CbAssociationsList	Speichert die Assoziationen, die aus CodeBeamer ausgelesen, werden als Liste
CbAssociationsListElement	Element der Assoziationsliste
CbEdgesList	Speichert die Kanten als Liste
CbEdgesListElement	Element der Kantenliste
CbVertexList	Speichert die Artefakte aus CodeBeamer sowie die ihnen zugeordneten Knoten als Liste
CbVertexListElement	Element der Knotenliste

Paket `de.unimannheim.wifo1.travis.utils`	
Klasse	**Beschreibung**
DirectionDisplayPredicate	Zeigt Pfeilspitzen der gerichteten Kanten an
EdgePaintFunctionImpl	Färbt die Kanten
RationaleTreeNode	Knotenelement des Baums, der die Rationale-Informationen darstellt
RationaleTreeRenderer	Legt das Layout des Rationale-Baums fest
ToolTipps	Zeigt die ToolTipps für die Knoten und die Kanten an
TraVisModalMouseGraph	Implementiert einige Funktionen für das Verhalten des Graphen. Erweitert die von JUNG zur Verfügung gestellten Klassen PluggableGraphMouse und ModalGraphMouse
VertexPaintFunktionImpl	Färbt die Knoten
VertexShapeFunctionImpl	Legt die Form der Knoten fest
VertexSizeFunctionImpl	Legt die Größe der Knoten fest
VertexStringerImpl	Beschriftet die Knoten

Tabelle 4.3: Beschreibung der Java Klassen

4.3 Implementierungsdetails

4.3.1 Datenstruktur

In TraVis sollen die verschiedenen Artefakte, die in der Kollaborationsplattform gespeichert sind, dargestellt werden. Zusätzlich sollen alle Abhängigkeitsbeziehungen der Artefakte untereinander darstellbar sein. Damit dies umgesetzt werden kann, müssen die Daten zunächst aus CodeBeamer geladen werden. Um

auf diese Daten schnell zugreifen zu können, müssen sie in einer eigenen Datenstruktur gespeichert werden. Für TraVis wurden dafür Doppelt-Verkettete Listen verwendet. Deren Bestandteile sollen in diesem Abschnitt erläutert werden. Zum grundsätzlichen Aufbau einer Doppelt-Verketteten Liste sei hier auf Cormen u. a. (2004) verwiesen.

Die verwendeten Listen werden im Paket `de.unimannheim.wifo1.travis.lists` implementiert. Ein Klassendiagramm dieses Paketes findet sich in Abbildung 7.2 auf Seite 34.

Insgesamt werden in TraVis drei verschiedene Listen eingesetzt: Eine `CbVertexList`, in der die Knoten gespeichert werden, eine `CbEdgesList`, in der die Kanten gespeichert werden, sowie eine `CbAssocationsList`, welche als Hilfsliste dient. Die drei Listen erben einen Teil ihres Verhaltens jeweils von der abstrakten Liste `AbstractList`.

CodeBeamer speichert alle Artefakte als *Dto-Objekte (Dto steht für Data Transfer Object), im folgenden Beispiel handelt es sich um Tracker-Objekte. Diese werden mit Hilfe von sogenannten finder-Methoden (im Beispiel findTrackersByProject), welche von der RemoteApi von CodeBeamer zur Verfügung gestellt werden, ausgelesen. Die Tracker-Objekte werden in einem Array gespeichert. Daraufhin wird die Methode createVertices aufgerufen, als Parameter werden dabei das soeben erzeugte Array und ein String, der die Art des Objektes speichert, übergeben.
Die folgenden Codeabschnitte sollen exemplarisch die Erzeugung der CbVertexList zeigen:

```
TrackerDto[] trackers = api.findTrackersByProject(token,
  projectId);
createVertices(trackers, "TrackerDto");

public void createVertices(Object[] obj, String kindOf) {
 for (int i = 0; i < obj.length; i++) {
  cbVertexList.add(new SparseVertex(), kindOf, obj[i], 20);
 }
}
```

Die Methode `createVertices()` besteht im Wesentlichen aus einer Schleife, die durch das Array läuft und für jedes Objekt ein neues Element an die CbVertexList hängt. Beim Hinzufügen wird ein neuer Knoten (`SparseVertex`) erzeugt. Dieser dient später zur Darstellung des Objektes in dem Graphen, der mit Hilfe des JUNG-Frameworks erstellt wird.

Der Code von CbVertexList.add ist wie folgt:

```
public void add(Vertex v, String kindOf, Object object) {
 length++;
 CbVertexListElement newElement = new CbVertexListElement(v,
   kindOf, object);
```

```
 newElement.next = first;
 newElement.previous = null;
 first = newElement;
}
```

Es wird also ein neues `CbVertexListElement` erzeugt, welches aus dem soeben erzeugten JUNG-Knoten sowie dem DTO aus CodeBeamer besteht. Zusätzlich wird in dem String kindOf gespeichert, welcher Art das Objekt ist. Diese Semantik hilft, das Objekt wiederzufinden.

Der Aufbau der beiden Klassen CbVertexList und CbVertexListElement ist in Abbildung 4.2 dargestellt.

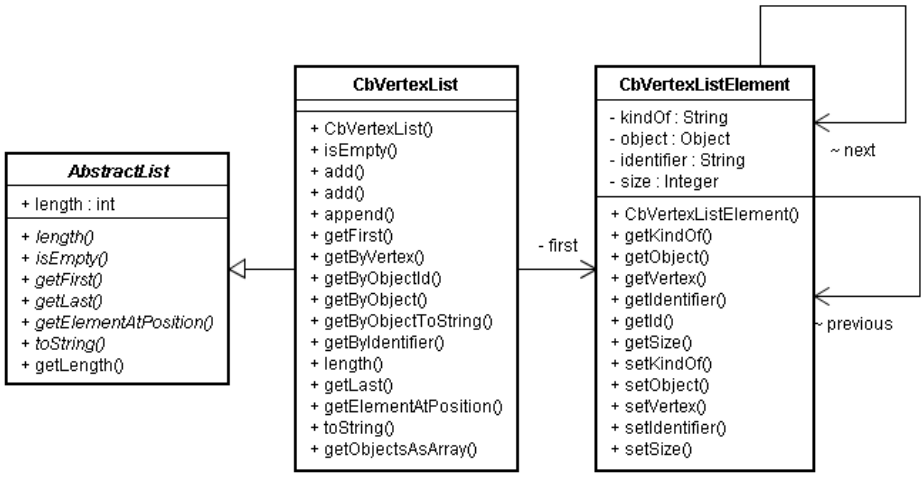

Abbildung 4.2: Aufbau der CbVertexList

Nachdem die Artefakte aus CodeBeamer ausgelesen und die `CbVertexList` gefüllt wurde, wird die `CbAssociationsList` erzeugt. Diese dient als Hilfsliste, in ihr werden lediglich die Assoziationen, die in CodeBeamer als `AssociationDto` vorhanden sind, gespeichert.

Aus `CbVertexList` und `CbAssociationsList` wird im Anschluss eine neue Liste, die `CbEdgesList` generiert. Der grundsätzliche Aufbau dieser Liste entspricht wiederum dem der `CbVertexList`.

In dieser Liste werden alle Kanten, die von TraVis dargestellt werden können, gespeichert. Es gibt dabei drei verschiedene Fälle, die bei der Erzeugung von Kanten berücksichtigt werden müssen: Zum einen gibt es Assoziationen, die wie im vorherigen Absatz erwähnt, als `AssociationDto` vorhanden sind. Diese können direkt in eine Kante umgewandelt werden. Zum anderen gibt es Assoziationen, die nicht direkt als Assoziation gekennzeichnet sind, aber zur Umsetzung größtmöglicher Nachvollziehbarkeit in TraVis dargestellt werden sollen. Dies sind beispielsweise Verbindungen zwischen einer Wiki-Seite und dem Nutzer, der diese Seite erstellt hat, oder Verbindungen zwischen einem TrackerItem und einem dazugehörenden Kommentar. Die dritte Art von Assoziation kann

mit Hilfe der Wiki-Syntax im Kommentar oder in der Beschreibung eines Artefakts vorkommen. Diese Art von Assoziation wird von TraVis allerdings noch nicht unterstützt, da die API der verwendeten CodeBeamer-Version 4.1.1 noch nicht die dazu erforderliche Funktionalität liefert.

Zwei so miteinander assoziierte CodeBeamer-Objekte sowie ein von JUNG zur Verfügung gestelltes Edge-Objekt werden in der `CbVertexList` gespeichert. Zusätzlich wird hier wieder ein String als semantische Hilfe hinzugefügt, in dem die Art der Assoziation vermerkt wird, damit die Kante leichter gefunden werden kann.

Die Notwendigkeit, sowohl die CodeBeamer-Objekte als auch die Knoten- oder Kanten-Objekte von JUNG in einer gemeinsamen Datenstruktur zu speichern ergibt sich daraus, dass es möglich sein muss, von einem dieser Objekte auf das andere zu schließen.
Beispielsweise soll es die Möglichkeit geben, alle Objekte einer Art, etwa Tracker, anzeigen zu lassen. Dazu wird die Liste, in der die KnotenElemente gespeichert sind, nach der entsprechenden Objektart (in diesem Fall `TrackerDto`) durchsucht und der entsprechende Knoten angezeigt.
Wird nun ein Knoten im Graphen ausgewählt, so sollen alle Informationen, die zu diesem Knoten verfügbar sind, angezeigt werden. Dazu wird wiederum die Liste durchsucht, diesmal jedoch nach dem Knoten-Element. Hier wird dann das entsprechende DTO zurückgegeben, aus dem die Informationen ausgelesen werden können.

4.3.2 Algorithmus für die wertbasierte Knotengrößenberechnung

Das Werkzeug soll die Möglichkeit bieten, Knoten entweder in der gleichen oder in verschiedenen Größen anzuzeigen. Bei der zweiten Darstellungsmöglichkeit soll sich die Größe der Knoten aus ihrem Wert ergeben (vergleiche Abschnitt 3.1). Standardmäßig werden die Knoten gleichrangig dargestellt. Zwischen den beiden Darstellungsmodi kann unter dem Menüpunkt “Vertexsize” im Menü “Options” umgeschaltet werden.

Die Methode, die die verschiedenen Knotengrößen berechnet, heißt `calculate VertexSize` und befindet sich in der Klasse `TraVisMainScreen`. Als Minimalgröße wurde der Wert 10 festgelegt. Die Größe berechnet sich dabei nach folgendem Schema:

TrackerItems, die Anforderungen repräsentieren, bekommen eine Größe zugewiesen, welche der Wichtigkeit entspricht, die mittels “*ibere*” ermittelt wurde und im CodeBeamer gespeichert ist. Die Formel, mit der sich die Größe des Knoten berechnet, lautet: `10 + value * 25`. Die Variable value wurde aus CodeBeamer ausgelesen und kann Werte zwischen 0 und 1 annehmen. Zum Minimalwert 10 wird also ein Wert zwischen 0 und 25 addiert, so dass den Anforderungsknoten Größen im Intervall [10,35] zugewiesen werden.

Die Größe der TrackerItems, die direkt mit Anforderungen verbunden sind, berechnet sich aus der Summe der assoziierten Anforderungen. Dabei wird vom

Wert jedes assoziierten Knotens zunächst 10 abgezogen, danach werden die Werte aufaddiert und schließlich wird wieder 10 hinzuaddiert, um eine einheitliche Darstellung zu gewährleisten. Als Maximalgröße wurde ein Wert von 60 festgelegt, da ansonsten Knoten, die mit mehreren Anforderungen assoziiert sind, unverhältnismäßig groß dargestellt werden müssten.

Die TrackerItems, die weder mit Anforderderungen verbunden sind noch selbst Anforderungen repräsentieren, bekommen eine Größe zugewiesen, die der in CodeBeamer gespeicherten Priorität entspricht. Diese Priorität kann Werte zwischen 1 und 5 annehmen, daher lautet diese Formel `10 + priority * 5`. Somit können auch diese Knoten eine maximale Größe von 35 bekommen.

Im Sinne des VBSE (vgl. Abschnitt 2.3) darf es keine Artefakte - im Sinne von Arbeitserzeugnissen, beispielsweise UML-Diagramme oder Quellcode - geben, die nicht direkt oder indirekt mit Anforderungen verbunden sind. Dadurch soll gewährleistet werden, dass nur für die Arbeitserzeugnisse Zeit aufgewendet wird, die einen Nutzen für den Kunden bringen.

Dementsprechend bekommen Dokumente, die im Dokumentenmanagementsystem von CodeBeamer gespeichert sind, eine Größe zugewiesen, die der Summe der eingehenden Kanten von TrackerItems entspricht. Ist dem Dokument kein TrackerItem zugewiesen, so bekommt es die minimale Größe von 10. Dies ist nach dem vorhergehenden Absatz jedoch nicht zulässig.

Gleiches gilt für die Quelldateien, die in einem eingebundenen Versionierungssystem (beispielsweise Subversion) eingebunden sind. Mit CodeBeamer 4.1.1 ist es jedoch noch nicht möglich, die Assoziationen zwischen TrackerItems und Quelldateien auszulesen.

Alle anderen Artefakte, wie beispielsweise Wiki-Seiten oder Forenbeiträge, liefern keinen direkten Mehrwert für den Kunden, sondern dienen lediglich Diskussions- oder Dokumentationszwecken. Daher bekommen alle Knoten, die weder TrackerItem noch Dokument oder Quelldatei sind, die minimale Größe von 10 zugewiesen.

4.3.3 Aufbereitung der Rationale-Informationen

Zur besseren Nachvollziehbarkeit sollen alle Informationen, die zu einem Artefakt verfügbar sind, dargestellt werden. Dadurch sollen die Gedanken des Rationale-Managements (vergleiche Abschnitt 2.2) und der Traceability (vergleiche Abschnitt 2.1) umgesetzt werden.

Dies geschieht zum einen dadurch, dass zwischen den Artefakten, die miteinander verknüpft sind, Kanten eingezeichnet werden, welche die Beziehung zwischen den Artefakten verdeutlichen. Zum anderen werden die Informationen in einem Baum dargestellt. Zusätzlich werden einige Informationen in ToolTipps angezeigt, welche über den Knoten beziehungsweise Kanten angezeigt werden.

Die konkrete Umsetzung dieser Anforderungen findet sich in der Methode `getRationaleVertex` in der Klasse `TraVisMainScreen` sowie in den Klassen `Tool-`

`Tipps`, `RationaleTreeNode` und `RationaleInformationScreen`. Das Aussehen des Graphen wird in der Klasse `RationaleTreeRenderer` festgelegt.

Für jeden Knoten im Graphen kann ein Rationale-Baum generiert werden, der angezeigt wird, sobald der Knoten ausgewählt wird. In den Rationale-Baum werden die Informationen zu den Artefakten angehängt, indem für jede Art von Information ein neues Blatt in den Baum gehängt wird. Die Blätter werden als `RationaleTreeNode` gespeichert. Ein `RationaleTreeNode` besteht aus einem String, mit dem das Blatt beschriftet wird, dem Objekt, das vom Knoten repräsentiert wird, einem String, der dem Blatt eine Semantik gibt - beispielsweise "TrackerItemDto" um zu zeigen, dass es sich um ein TrackerItem handelt - sowie einer Boole'schen Variable, die anzeigt, ob es sich um ein Informationselement oder um ein assoziiertes Element handelt. Diese Variable ist jedoch nur für das Aussehen des Blattes von Bedeutung.

Als oberstes Element wird jeweils ein Informationselement angezeigt. Wird es ausgewählt, dann werden alle Informationen, die zu dem Artefakt verfügbar sind, in einem neuen Fenster angezeigt. Der Umfang der Informationen hängt dabei sowohl vom Artefakttyp - also ob es sich beispielsweise um ein TrackerItem, Dokument usw. handelt - als auch vom individuellen Inhalt des Artefakts ab.

Falls der Knoten ein TrackerItem, ein Dokument oder eine Quelldatei ist, dann werden alle Artefakte, die mit dem Knoten direkt verbunden sind, zusätzlich in dem Baum dargestellt. Dabei werden sie in gesonderte Kategorien eingeteilt. Kommentare zu TrackerItems finden sich beispielsweise im Ast "TrackerItem-Comments", Attachments von TrackerItems dagegen im Ast "Attachments".

Bei Artefakten, die keine besondere Nachvollziebarkeit erfordern, wird lediglich das Informationselement und keine weiteren assoziierten Knoten angezeigt. Dies ist beispielsweise bei jeder Art von Kommentaren der Fall.

Die Klasse `RationaleInformationScreen` ist für die Darstellung der Informationen zuständig. Wenn ein RationaleTreeNode-Objekt in dem Baum ausgewählt wird, dann wird eine Instanz dieser Klasse erzeugt, dabei wird das RationaleTreeNode-Objekt an den Konstruktor übergeben. Im Konstruktor wird das Aussehen des neuen Fensters festgelegt, außerdem wird die Methode `displayInformation` aufgerufen. In dieser Methode wird überprüft, welcher Art das Objekt in dem Knoten ist. Entsprechend dem Typ des Objekts wird eine weitere Methode aufgerufen, die sich um die Informationen kümmert, die dargestellt werden. Für TrackerItems ist das beispielsweise die Methode `getTrackerItemInformation`, die einen String zurückgibt. Die Methode `getTrackerItemInformation` liest alle Felder aus dem TrackerItem aus und fügt diese in den String ein, sofern sie nicht auf null gesetzt sind.

Die Klasse `ToolTipps` ist dagegen für die ToolTipps über den Knoten und Kanten zuständig. In den ToolTipps werden ebenfalls einige Informationen angezeigt, diese sollen jedoch nur einen kurzen Überblick über das Objekt geben und sind daher nicht so umfangreich wie die Informationen im Rationale-Baum.

4.3.4 Visualisierung mit JUNG

Zur Visualisierung der Abhängigkeitsbeziehungen wird für TraVis die Open-Source Bibliothek JUNG verwendet (siehe Kapitel 4.1.2).

Die wichtigsten Bestandteile von JUNG sind der Graph, welcher aus Knoten und Kanten besteht, sowie die beiden Klassen `PluggableRenderer` und `VisualizationViewer`. TraVis erstellt den Graphen in der Methode `loadProject` in der Klasse `TraVisMainScreen`. Der folgende Codeausschnitt zeigt diesen Vorgang exemplarisch:

```
g = new SparseGraph();
graphLayout = new CircleLayout(g);
pr = new PluggableRenderer();
vv = new VisualizationViewer(graphLayout, pr);
```

Zunächst wird der Graph erzeugt, diesem wird anschließend ein Layout zugewiesen. Dazu stehen in TraVis insgesamt drei verschiedene Layouts zur Verfügung: `CircleLayout`, `ChaosLayout`, und `StaticLayout`. Alle drei Layout-Klassen erben von der Klasse `AbstractLayout` und befinden sich im Paket `de.unimannheim.wifo1.travis.layout`.

Danach wird eine Instanz der Klasse `PluggableRenderer` erzeugt. Durch den `PluggableRenderer` kann das Design des Graphen festgelegt werden. So kann man ihm mittels seiner setter-Methoden Klassen zuweisen, welche beispielsweise den Graphen beschriften oder die Farben der Bestandteile ändern.
Der folgende Aufruf sorgt beispielsweise dafür, dass jeder Knoten mit dem richtigen Namen beschriftet wird:

```
pr.setVertexStringer(new VertexStringerImpl(cbVertexList));
```

Schließlich wird mit Hilfe des Layouts und des `PluggableRenderers` eine Instanz des `VisualizationViewers` erzeugt. Dieser ist allgemein für das Verhalten des Graphen zuständig, er verwaltet beispielsweis die verschiedenen Listener, um auf Ereignisse zu reagieren, oder die Zoomfunktion.

Nachdem das Aussehen und das Verhalten des Graphen festgelegt sind, müssen noch die Knoten und die Kanten hinzugefügt werden. Dies geschieht in TraVis mittels der Methoden `createVertices` und `createEdges`. `CreateVertices` erzeugt neue Knoten und fügt diese sofort zur `cbVertexList` hinzu, `createEdges` erzeugt entsprechend neue Kanten und fügt diese zur `CbEdgesList` hinzu. Der Code der Methode `createVertices` ist dabei wie folgt:

```
private void createVertices(Object[] obj, String kindOf) {
  for (int i = 0; i < obj.length; i++) {
    cbVertexList.add(new SparseVertex(), kindOf, obj[i], 20);
  }
}
```

5 Nutzungsbeschreibung

5.1 Installation

5.1.1 Voraussetzungen

Die vorliegende Software benötigt eine Java-Laufzeitumgebung ab Version 1.5.0 sowie als Datengrundlage einen CodeBeamer-Server ab Version 4.1.1.

5.1.2 Komponenten

Die Software enthält folgende Komponenten.

- Die Bibliothek cb-api der Firma Intland in Version 4.1.1. Diese stellt die Schnittstelle zur CodeBeamer-Plattform dar.
- Die Open-Source Bibliothek JUNG in Version 1.7.4, welche zur Visualisierung genutzt wird.
- Die von JUNG benötigten Open-Source Bibliotheken colt, commons-collections in Version 3.1 sowie commons-lang in Version 2.1.
- Die Open-Source Bibliothek jdom, welche für das XML-Parsing der Sprachdatei verwendet wird.

5.1.3 Installation und Start

Das ausführbare Archiv `travis-1.0.jar` aus dem Verzeichnis `TraVis/lib` der dieser Arbeit beiligenden CD muss in einen beliebigen Ordner der Festplatte kopiert werden. Ein Doppelklick auf das Programmsymbol oder die Eingabe von *java jar travis-1.0.jar* auf der Kommandozeile startet das Programm.

5.2 Anwendungsfälle

In diesem Abschnitt werden einige Anwendungsfälle von TraVis erläutert. Dabei wird das Projekt TraVis, mit dem die Anfertigung dieser Arbeit begleitet wird, als Beispielprojekt verwendet.

5.2.1 Einloggen und Projekt laden

Nach dem Start von TraVis (siehe Abschnitt 5.1.3) erscheint das in Abbildung 5.1 dargestellte Login-Fenster. Im Feld "URL" muss die URL der API des CodeBeamer-Servers angegeben werden, für dieses Beispiel ist das `http://wifo1- 52.bwl.uni-mannheim.de:8080/cb/remote-api`. Im Feld "Name" muss der Username des Benutzers eingegeben werden, im Feld "Password" entsprechend das Passwort des Benutzers. Unter "Language" kann zwischen verschiedenen Sprachprofilen gewählt werden. In der für diese Arbeit entwickelten Version 1.0 stehen ein deutsches und ein englisches Sprachprofil zur Verfügung. Im Folgenden wird das englische Profil verwendet.

Abbildung 5.1: Login-Fenster

Nach dem Einloggen erscheint das Hauptfenster von TraVis.

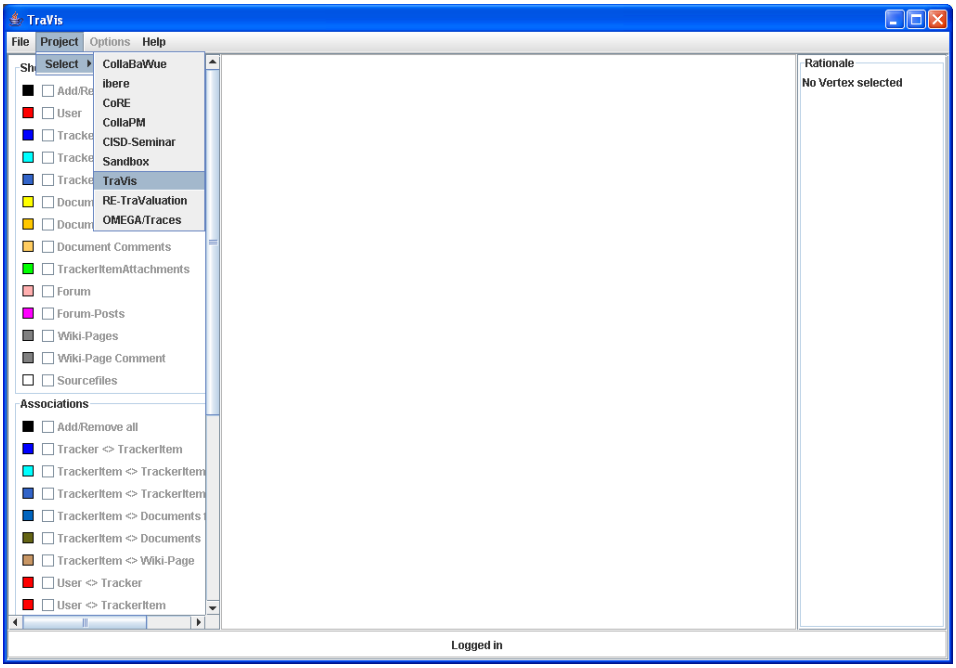

Abbildung 5.2: Projektauswahl

Im Untermenü “Select” im Menü “Project” werden alle Projekte angezeigt, in denen der Benutzer angemeldet ist (siehe Abbildung 5.2). Wählt man dort ein Projekt aus, dann wird dieses geladen (vergleiche 4.3.1). Dieser Vorgang kann in Abhängigkeit von der Größe des Projekts einige Zeit in Anspruch nehmen. Sobald das Projekt vollständig geladen ist, werden das Optionenmenü und die Checkboxen auf der linken Seite freigeschaltet.

5.2.2 Knoten und Kanten anzeigen

Mit Hilfe der Checkboxen auf der linken Seite können die geladenen CodeBeamer-Artefakte als Knoten dargestellt werden. Dabei können entweder alle Knoten auf einmal oder einzelne Kategorien, beispielsweise Dokumente, angezeigt werden.
Die Assoziationen zwischen den Artefakten können ebenfalls über Checkboxen aktiviert werden. Abbildung 5.3 zeigt eine solche Ansicht exemplarisch.

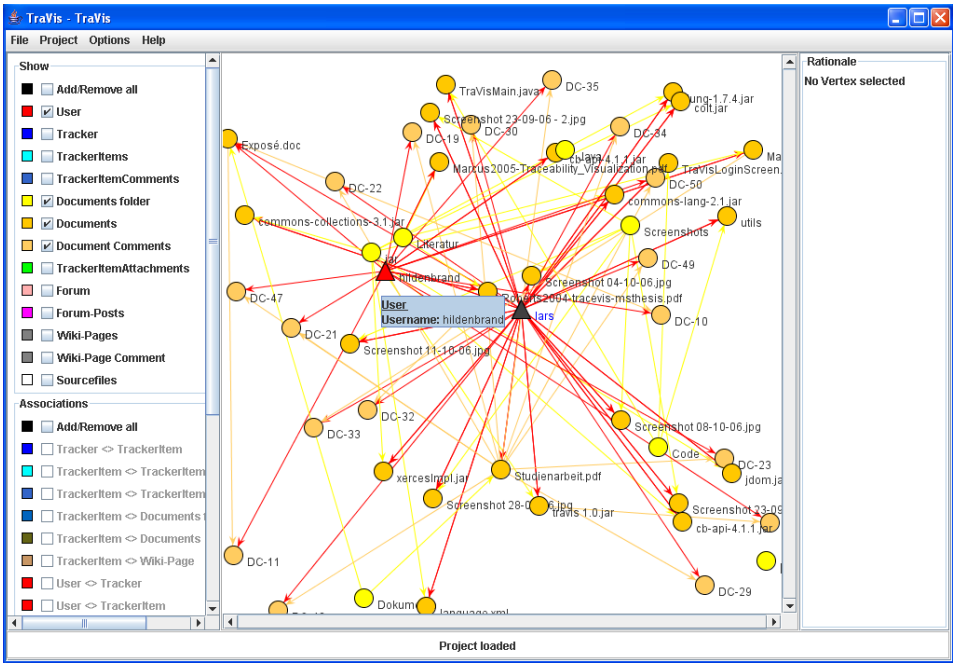

Abbildung 5.3: Dokumente anzeigen lassen

5.2.3 Darstellung ändern

In TraVis werden insgesamt drei verschiedene Layouts unterstützt, zwischen denen jederzeit umgeschaltet werden kann. Die Umschaltung zwischen den Layouts kann unter dem Menüpunkt “Representations” im Menü “Options” erfolgen. Zur Verfügung stehen das Circle-Layout, das Chaos-Layout und das ISOM-Layout. Das Circle-Layout ordnet alle Artefakte in einem Kreis an, das Chaos-Layout belässt die Artefakte ungeordnet und das ISOM-Layout sorgt für eine isometrische Darstellung.

5.2.4 Knotengröße ändern

Die Größe der Knoten kann ebenfalls geändert werden. Dazu stehen im Untermenü “Vertexsize” im Menü “Options” zwei Möglichkeiten zur Verfügung: Zum einen können die Knoten einheitlich dargestellt werden, das heisst, dass alle Knoten die gleiche Größe haben, zum anderen können die Knoten eine Größe haben, die ihrem Wert entspricht. Wie sich diese Größe berechnet ist in Abschnitt 4.3.2 beschrieben. Abbildung 5.4 zeigt die Vorgehensweise für das Umschalten der Knotengröße.

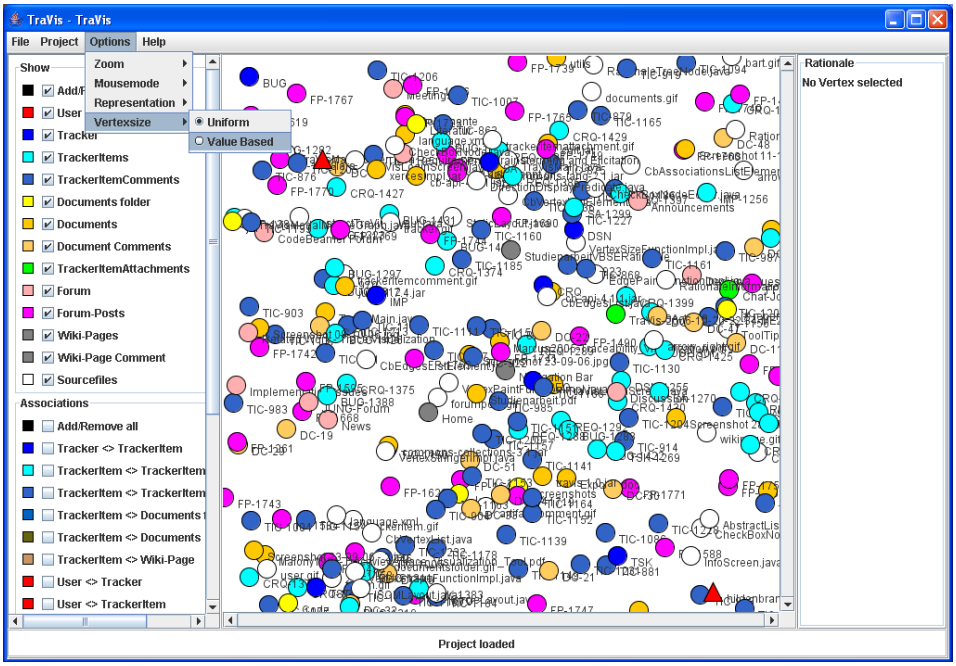

Abbildung 5.4: Knotengröße verändern

5.2.5 Mit Knoten assoziierte Knoten anzeigen

Durch die Wertbasierte Darstellung ist es möglich, die wichtigen Artefakte auf einen Blick zu erkennen. Da die Darstellung speziell bei großen Projekten sehr unübersichtlich werden kann, gibt es die Möglichkeit, nur Artefakte anzeigen zu lassen, die mit einem bestimmten Artefakt verbunden sind. Durch einen Rechtsklick auf einen Knoten kann das Kontextmenü zu diesem erreicht werden. Sobald dort “Show connected Vertices” ausgewählt wird, werden nur die entsprechenden Artefakte angezeigt. Abbildung 5.5 zeigt die Vorgehensweise und Abbildung 5.6 das Ergebnis der Auswahl.

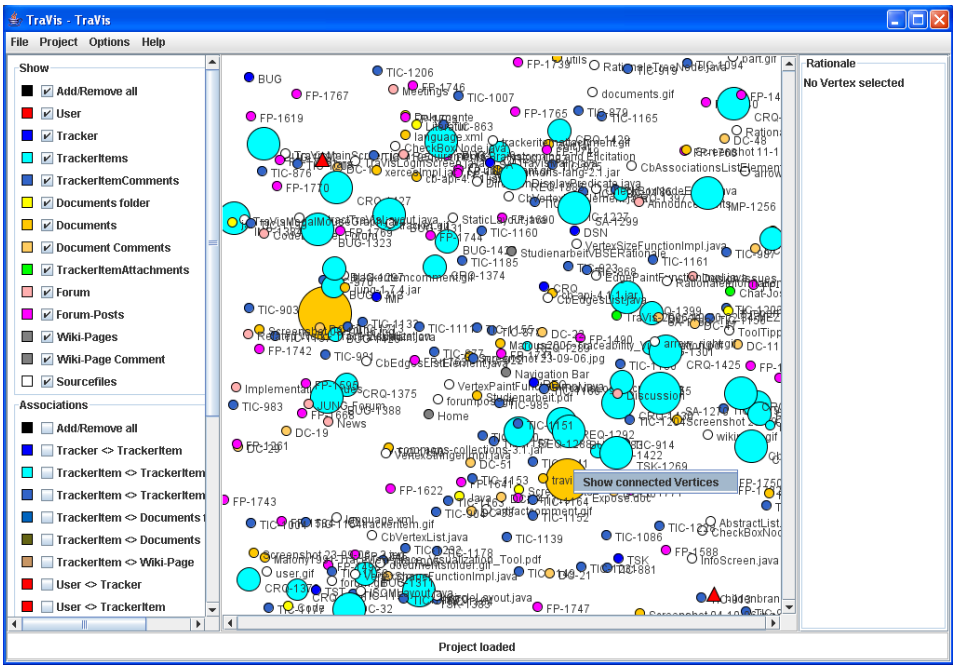

Abbildung 5.5: Kontextmenü

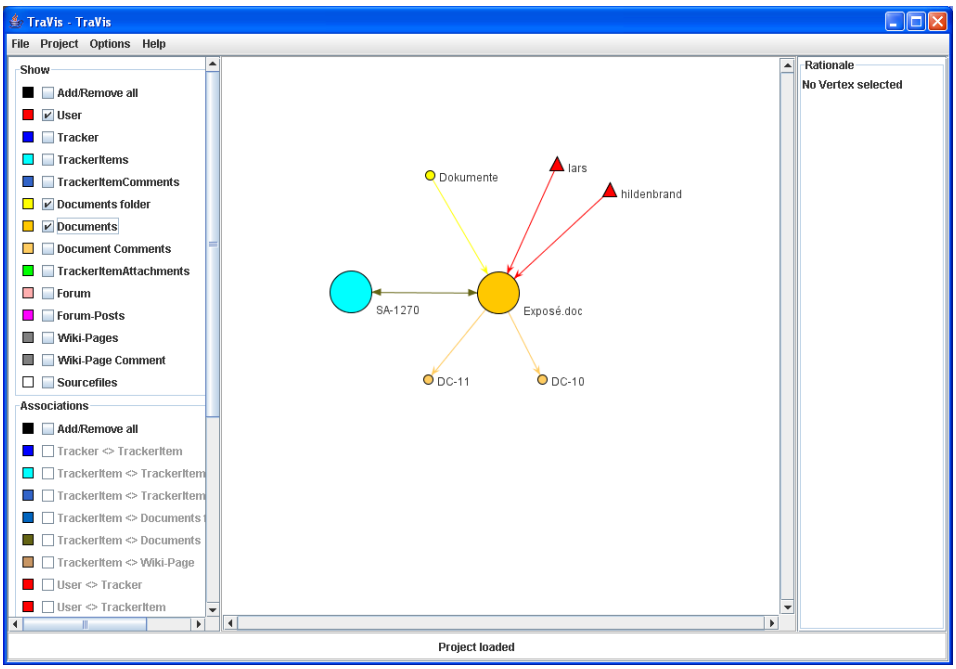

Abbildung 5.6: Nur assoziierte Knoten anzeigen

5.2.6 Rationale-Baum anzeigen

Wenn mit der linken Maustaste ein Knoten ausgewählt wird, dann wird der Baum mit den Rationale-Informationen zu diesem Artefakt angezeigt. Das Schema, nach dem der Baum erzeugt wird, ist in Abschnitt 4.3.3 beschrieben. Der Rationale-Baum für das Dokument Exposé.doc ist in Abbildung 5.7 dargestellt.

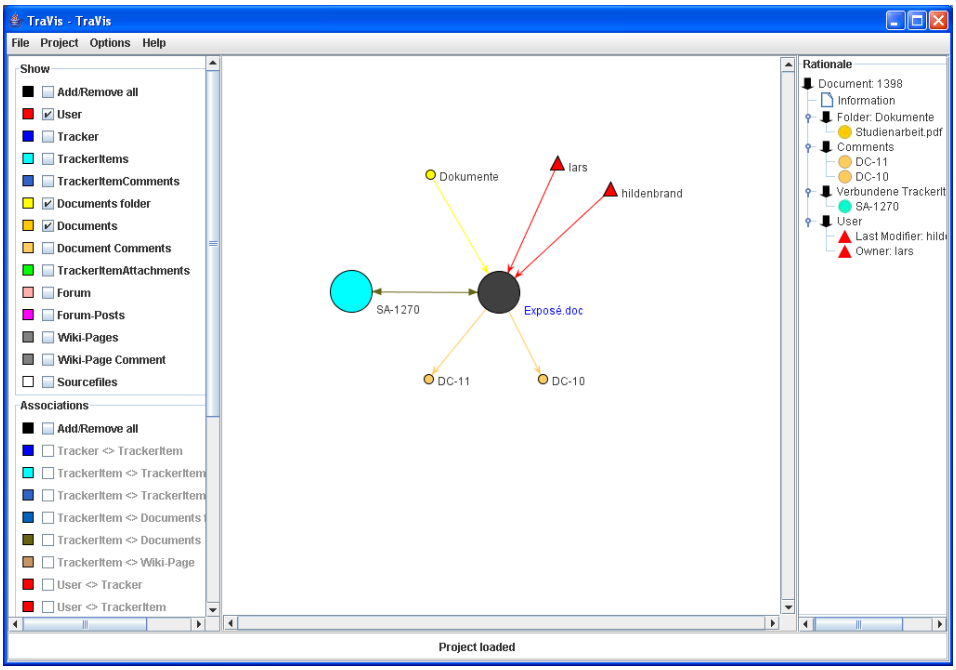

Abbildung 5.7: Rationale-Baum zu einem ausgewählten Artefakt

Wie bereits in Abschnitt 4.3.3 beschrieben wurde, wird ein neues Fenster geöffnet, wenn ein Blatt in dem Rationale-Baum ausgewählt wird. Ein solches Fenster zeigt Abbildung 5.8 exemplarisch.

Abbildung 5.8: Rationale-Fenster

6 Zusammenfassung und Ausblick

Das entwickelte Werkzeug ermöglicht es dem Benutzer, Daten aus einem Projekt aus der Kollaborationsplattform CodeBeamer auszulesen und mittels eines Graphen zu visualisieren. Dafür stehen verschiedene Layouts zur Verfügung. Die Knoten, welche die Daten aus der Plattform repräsentieren, können gemäß den Gedanken des in Abschnitt 2.3 vorgestellten VBSE ihre dargestellte Größe ändern, um ihre Wertigkeit anzuzeigen. Außerdem können die Rationale-Informationen (vgl. Abschnitt 2.2) für jeden Knoten angezeigt werden. Durch die Auswertung und Darstellung der Assoziation zwischen den Knoten wird das in Abschnitt 2.1 vorgestellte Nachvollziehbarkeitsmanagement umgesetzt.

Die Darstellung der Rationale-Informationen umfasst alle Informationen, die von der API von CodeBeamer 4.1.1 zur Verfügung gestellt werden. Gleiches gilt für die Assoziationen zwischen den Artefakten. In zukünftigen Versionen von TraVis ist es denkbar, dass auch die Assoziationen, die sich aus Verknüpfungen mittels der Wiki-Syntax ergeben, analysiert werden. Gleiches gilt für projektübergreifende Assoziationen, welche zwar in der Kollaborationsplattform erzeugt werden können, auf die aber über die API nicht zugegriffen werden kann.

In der vorliegenden Version des Werkzeugs können die Knoten entweder einheitlich oder wertbasiert dargestellt werden. Möglich wären jedoch auch weitere Algorithmen, um die Knotengröße zu bestimmen, wie zum Beispiel nach der Anzahl der eingehenden beziehungsweise ausgehenden Kanten.

Eine sinnvolle Erweiterung des Werkzeugs wäre zudem die Möglichkeit, neue Assoziationen zwischen den Knoten zu erzeugen und diese in der Kollaborationsplattform zu speichern. Ebenfalls denkbar wären verschiedene Sichtweisen auf den Graphen: beispielsweise könnten weniger wichtige Knoten ausgeblendet oder nur das soziale Netzwerk zwischen den Nutzern angezeigt werden. Dazu könnte analysiert werden, welche User über welche Artefakte miteinander in Beziehung stehen.

Denkbar wären auch verschiedene vordefinierte Filter, nach denen rollenbasierte Sichtweisen definiert werden können: Für Projektmanager sind möglicherweise andere Informationen interessant als für Entwickler.

Eine bessere Integration in die Kollaborationsplattform CodeBeamer könnte durch die Portierung des Werkzeugs in eine Webanwendung erfolgen. Dafür würde sich Java Webstart[1] anbieten.

[1] http://java.sun.com/products/javawebstart (5.10.2006)

7 Anhang

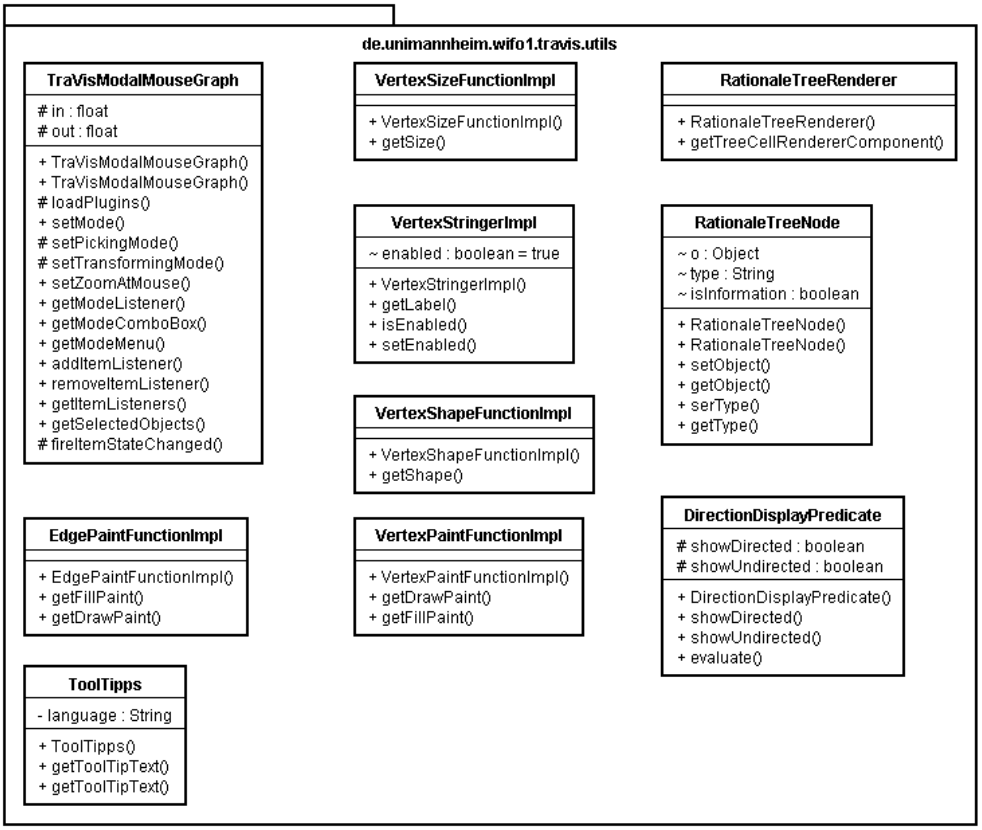

Abbildung 7.1: Paket de.unimannheim.wifo1.travis.utils

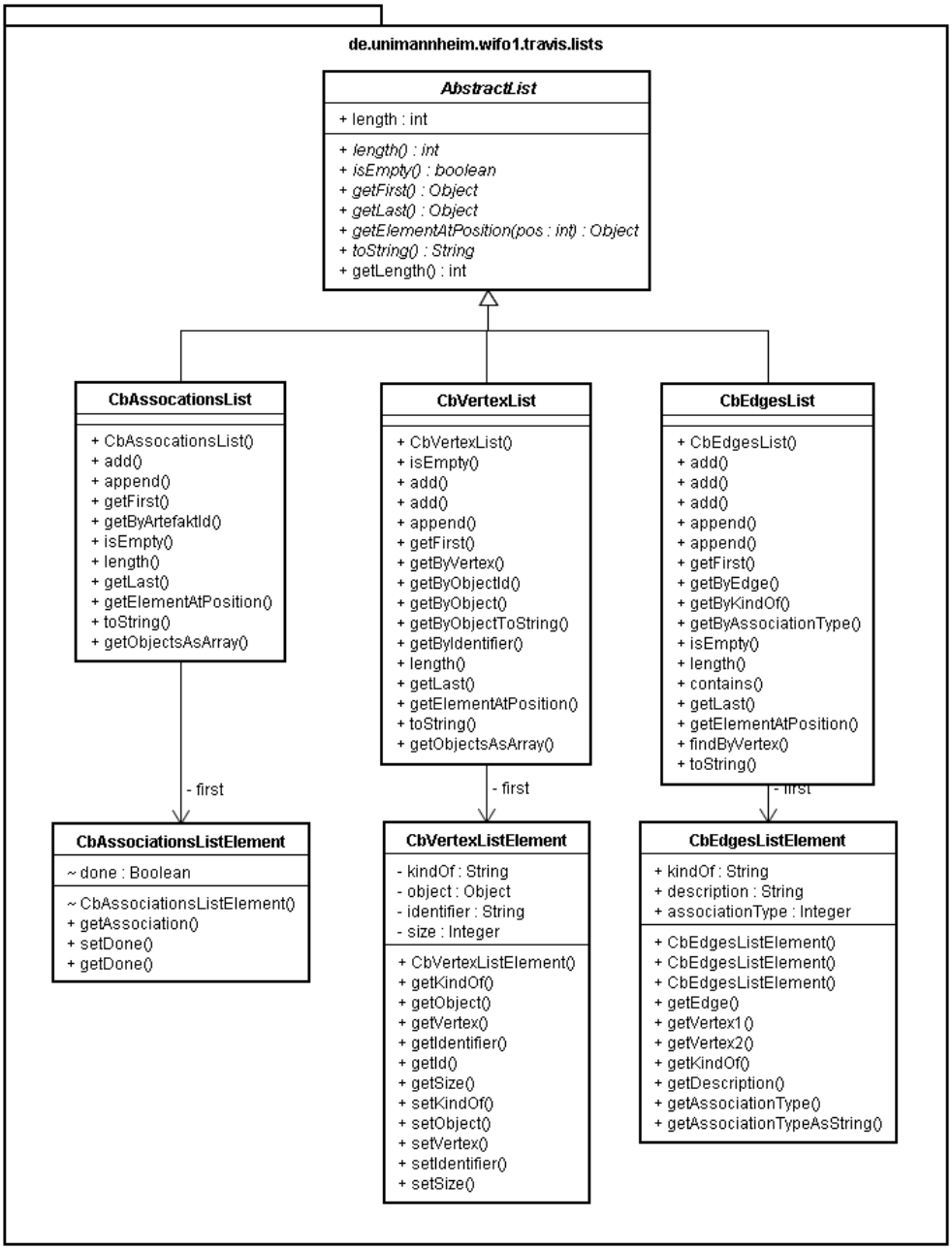

Abbildung 7.2: Paket de.unimannheim.wifo1.travis.lists

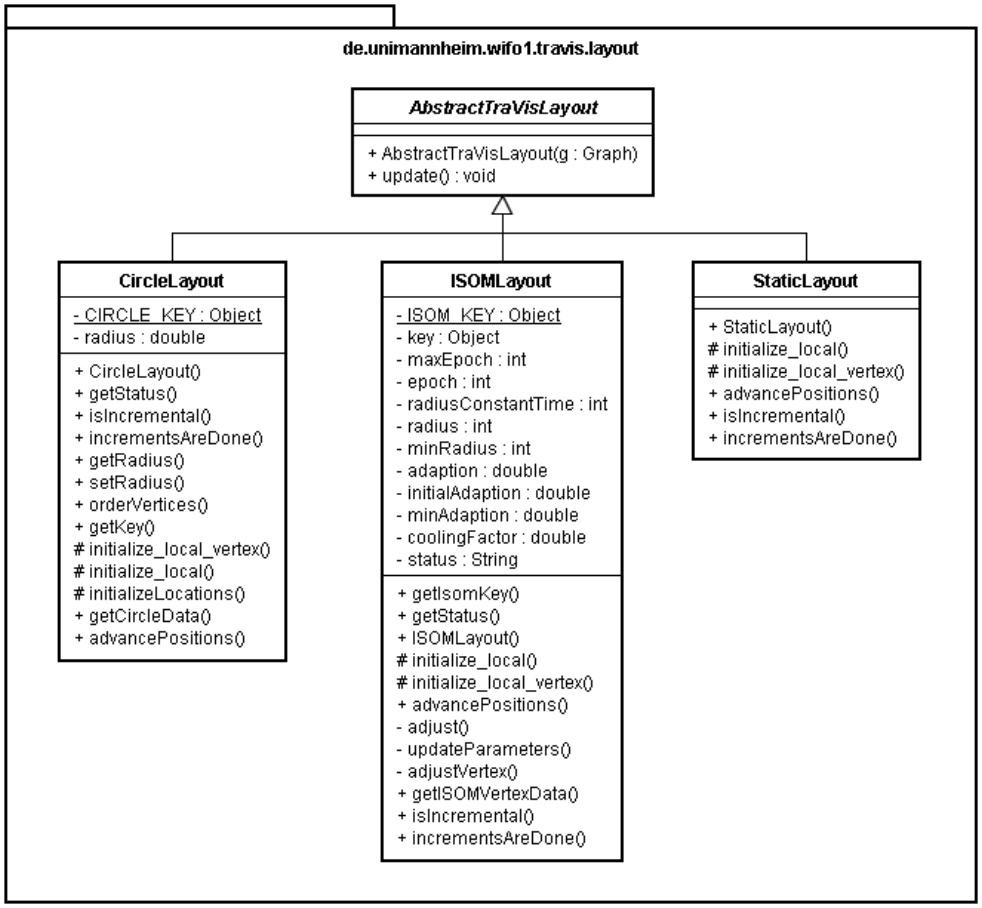

Abbildung 7.3: Paket de.unimannheim.wifo1.travis.layout

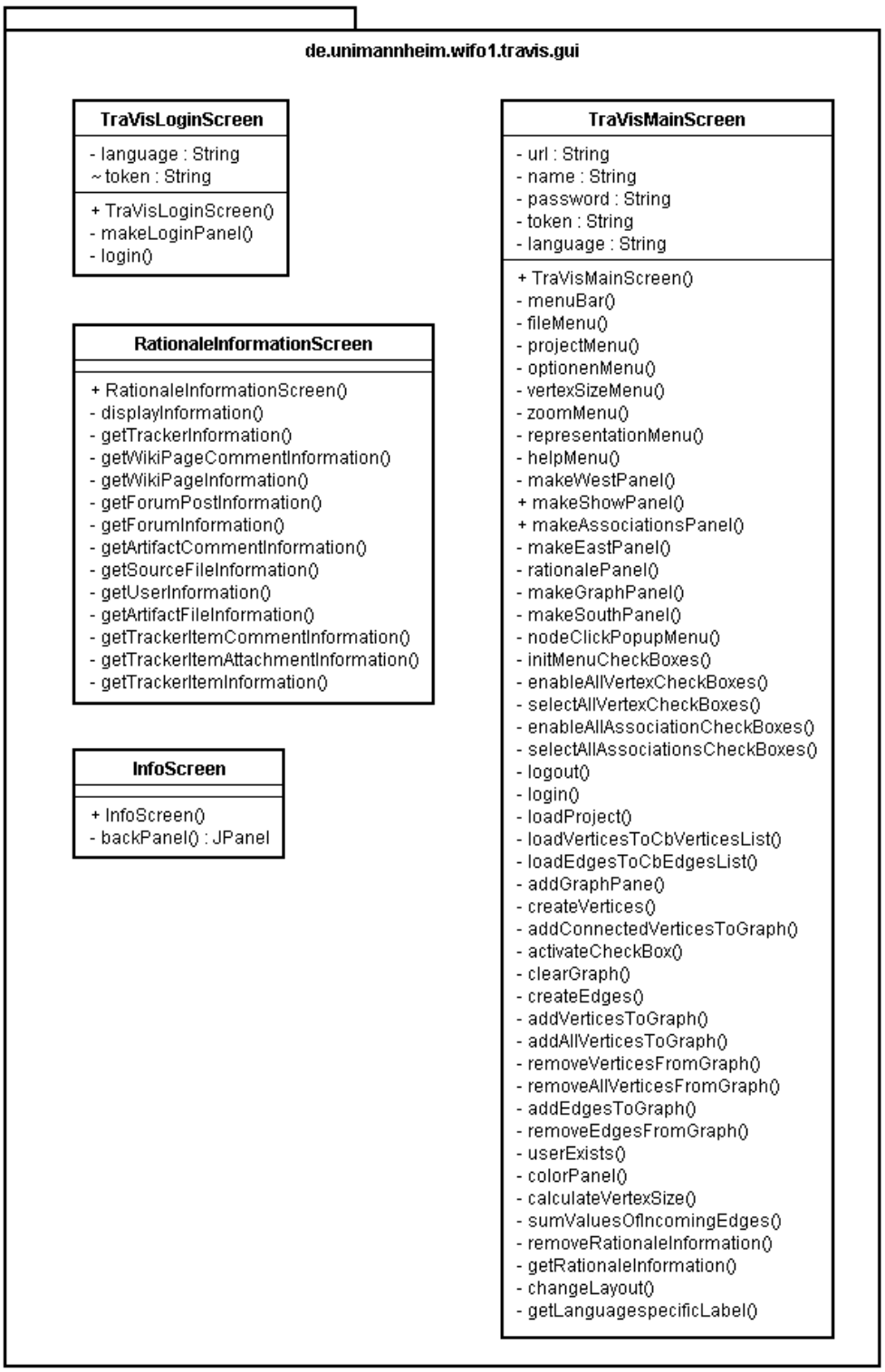

Abbildung 7.4: Paket de.unimannheim.wifo1.travis.gui

Literaturverzeichnis

[Antoniol u. a. 2002]
ANTONIOL, G. ; CANFORA, G. ; CASAZZA, G. ; DE LUCIA, A. ; MERLO, E.: Recovering Traceability Links between Code and Documentation. In: *IEEE Transactions on Software Engineering* 28 (2002), October, Nr. 10, S. 970–983

[Berry u. Aurum 2006]
BERRY, M. ; AURUM, A.: Measurement and Decision Making. In: BIFFL, S. (Hrsg.) ; AURUM, A. (Hrsg.) ; BOEHM, B. (Hrsg.) ; ERDOGU, H. (Hrsg.) ; GRÜNBACHER, P. (Hrsg.): *Value-Based Software Engineering.* Springer Verlag, 2006, Kapitel 8, S. 155–177

[Biffl u. a. 2006]
BIFFL, S. ; AURUM, A. ; BOEHM, B. ; ERDOGUS, H. ; GRÜNBACHER, P.: *Value-Based Software Engineering.* Springer Verlag, 2006

[Boehm 2003]
BOEHM, B.: Value-Based Software Engineering. In: *Software Engineering Notes* 28 (2003), March, Nr. 2, S. 1–12

[Boehm 2006]
BOEHM, B.: Value-Based Software Engineering: Overview and Agenda. In: BIFFL, S. (Hrsg.) ; AURUM, A. (Hrsg.) ; BOEHM, B. (Hrsg.) ; ERDOGUS, H. (Hrsg.) ; GRÜNBACHER, P. (Hrsg.): *Value-Based Software Engineering.* Springer Verlag, 2006, Kapitel 1, S. 3–14

[CMMI 2002]
CMMI: *Capability aturity Model Integration (CMMI), Version 1.1.* Pittsburgh, USA: Software Engineering Institute, 2002

[Cormen u. a. 2004]
CORMEN, T.H. ; LEISERSON, C.E. ; RIVEST, R. ; STEIN, C.: *Algorithmen - Eine Einführung.* Oldenbourg Wissenschaftsverlag, 2004

[DOD-STD-2167A 1988]
DOD-STD-2167A: *Military Standard. Defense Systems Software Development.* Washington, DC., USA: Department of Defense, 1988

[Dutoit u. Paech 2001]
DUTOIT, A. ; PAECH, B.: Rationale Management in Software Engineering. In: SK, Chang (Hrsg.): *Handbook on Software Engineering and Knowledge Engineering* Bd. 1. World Scientific, 2001

[Dutoit u. a. 2006]
DUTOIT, A.H. ; MCCALL, R. ; MISTRÍK, I. ; PAECH, B.: Rationale Management in SoftwareEngineering: Concepts and Techniques. In: DUTOIT, A.H. (Hrsg.) ; MCCALL, R. (Hrsg.) ; MISTRÍK, I. (Hrsg.) ; PAECH, B. (Hrsg.): *Rationale Management in Software Engineering*. Springer Verlag, 2006, Kapitel 1, S. 1–48

[Egyed 2006]
EGYED, A.: Tailoring Software Traceability to Value-Based Needs. In: BIFFL, S. (Hrsg.) ; AURUM, A. (Hrsg.) ; BOEHM, B. (Hrsg.) ; ERDOGUS, H. (Hrsg.) ; GRÜNBACHER, P. (Hrsg.): *Value-Based Software Engineering*. Springer Verlag, 2006, Kapitel 14, S. 287–308

[Egyed u. a. 2005]
EGYED, A. ; BIFFL, S. ; HEINDL, M. ; GRÜNBACHER, P.: 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*, ACM Press, 2005. – ISBN 1–59593–243–7, S. 2–7

[Geisser u. Hildenbrand 2006a]
GEISSER, M. ; HILDENBRAND, T.: Agiles, verteiltes Requirements-Engineering mit Wikis und einer kollaborativen Softwareentwicklungsplattform. In: *OBJEKTSpektrum* 6 (2006), S. 36–41

[Geisser u. Hildenbrand 2006b]
GEISSER, M. ; HILDENBRAND, T.: A Method for Collaborative Requirements Elicitation and Decision-Supported Requirements Analysis. In: OCHOA, S.F. (Hrsg.) ; ROMAN, G.-C. (Hrsg.): *Advanced Software Engineering: Expanding the Frontiers of Software Technology* Bd. 219. Springer, Boston, 2006, S. 108–122

[Geisser u. a. 2006]
GEISSER, M. ; HILDENBRAND, T. ; KLIMPKE, L. ; RASHID, A.: Werkzeuge zur kollaborativen Softwareerstellung - Stand der Technik / Universität Mannheim. 2006. – Forschungsbericht

[Gotel u. Finkelstein 1994]
GOTEL, O.C.Z. ; FINKELSTEIN, A.C.W.: An Analysis of the Requirements Traceability Problem. In: *1st International Conference on Requirements Engineering*, 1994, S. 94–101

[Heindl u. Biffl 2006]
HEINDL, Matthias ; BIFFL, Stefan: Risk management with enhanced tracing of requirements rationale in highly distributed projects. In: *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*. ACM Press. – ISBN 1–59593–404–9, 20–26

[Horner u. Atwood 2006]
HORNER, J. ; ATWOOD, M.: Effective Design Rationale: Understanding the

Barriers. In: Dutoit, A.H. (Hrsg.) ; McCall, R. (Hrsg.) ; Mistrík, I. (Hrsg.) ; Paech, B. (Hrsg.): *Rationale Management in Software Engineering*. Springer Verlag, 2006, Kapitel 3

[IEEE 1219 1992]
IEEE 1219: *IEEE standard for software maintenance*. New York, USA: Institute of Electrical and Electronic Engineers, 1992

[IEEE 610 1990]
IEEE 610: *IEEE Std 610 Standard Glossary of Software Engineering Terminology*. New York, USA: Institute of Electrical and Electronic Engineers, 1990

[ISO 9000-3 1991]
ISO 9000-3: *Quality management and quality assurance standards*. Geneve, Switzerland: International Organization for Standardization, 1991

[Lee 1997]
Lee, J.: Design Rationale Systems: Understanding the Issues. In: *IEEE Expert* 12 (1997), Nr. 3, S. 78–85

[Lindvall u. Sandahl 1996]
Lindvall, M. ; Sandahl, K.: Practical Imications of Traceability. In: *Software Practice and Experience* Bd. 26, 1996, S. 1161–1180

[Marcus u. a. 2005]
Marcus, A. ; Xie, X. ; Poshyvank, D.: When and How to Visualize Traceability Links? In: *TEFSE '05: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, ACM Press, 2005, S. 56–61

[Objective Systems SF AB 1993]
Objective Systems SF AB: *Objectory Analysis and Design 3.3 Tool*. 1993

[Sommerville 2007]
Sommerville, I.: *Software Engineering*. 8. Auflage. Addison Wesley, 2007

[de Souza u. a. 2004]
Souza, C. de ; Dourish, P. ; Redmiles, D. ; Quirk, S. ; Trainer, E.: From Technical Dependencies to Social Dependencies. In: *Social Networks Workshop at the CSCW Conference*. Chicago, 2004

[Standish Group 1994]
Standish Group: *Chaos Report*, 1994. `http://www.standishgroup.com/`

[SWEBOK 2004]
SWEBOK ; Abran, A. (Hrsg.) ; Bourque, P. (Hrsg.) ; Dupuis, R. (Hrsg.) ; Moore, J.W. (Hrsg.) ; Tripp, L.L. (Hrsg.): *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Computer Society, 2004

[Trainer u. a. 2005]
TRAINER, E. ; QUIRK, S. ; SOUZA, C.R.B. de ; REDMILES, D.F.: Bridging the Gap between Technical and Social Dependencies wirh Ariadne. In: *Proceedings of the Eclipse Technology eXchange (ETX) Workshop*. San Dieo, 2005

[VA Software 2006]
VA SOFTWARE: *SourceForge Enterprise Edition 4.3 Service Pack 1 User Guide*. 2006

[Virtuelles Software Engineering Kompetenzzentrum]
VIRTUELLES SOFTWARE ENGINEERING KOMPETENZZENTRUM: *Nachvollziehbarkeits-Beziehungen*. `http://www.software-kompetenz.de/?15419(20.08.2006)`