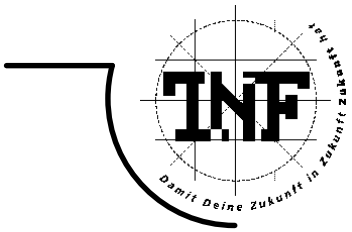
JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis



Hyperlinked Project Data for Software Engineering Experiments

Bakkalaureatsarbeit
(Projektpraktikum)

zur Erlangung des akademischen Grades

Bakkalaureus der Technischen Wissenschaften

in der Studienrichtung

Informatik

Angefertigt am *Institute for Systems Engineering and Automation*

Betreuung:

Univ.-Prof. Dr. Alexander Egyed, M.Sc.

Eingereicht von:

Thomas Schartmüller

Linz, März 2009

Kurzfassung

Trace links are essential for many software development and maintenance activites. Despite significant advances in traceability research, recovering trace links remains a human-intensive activity. Surprisingly little is known about this human factor [1]

So beschreiben Univ.-Prof. Dr. Alexander Egyed, M.Sc., Florian Graf und a.Univ.-Prof. Dr. Paul Grünbacher die Motivation hinter ihrem Projekt. Es galt herauszufinden, wie sich verschiedenste Faktoren auf die Korrektheit von Trace-Links auswirken.
Für einen kontrollierten Ablauf der Experimente wurde ein eigenentwickeltes Programm verwendet.

Im Zuge dieses Projekts sollte nun eine Neuentwicklung, in Anlehnung an das bestehende Programm, entstehen. Ziel war es, eine leichtere Bedienung, vor allem aber eine bessere Wartbarkeit und Plattformunabhängigkeit zu erreichen. Weiters sollte das neue Programm auch für weitere, eventuell andere Versuchsanordnungen gewappnet sein.

Das Resultat ist eine auf PHP basierende Webapplikation mit Datenbankanbindung, welche zentral verwaltet und mit Daten befüllt werden kann. Probleme mit der Browserkompatibilität wurden tunlichst vermieden bzw. gelöst und somit ist ein plattformunabhängiges Arbeiten gewährleistet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Motivation

1.1 Grundgedanke/Hintergründe

Diesem Projekt vorangegangen war ein Experiment von Univ.-Prof. Dr. Alexander Egyed, M.Sc., Florian Graf und a.Univ.-Prof. Dr. Paul Grünbacher, mit dem Titel „Recovering Trace Links between Requirements and Code: Understanding the Human Loop“, welches sich mit der Auffindung von Pfaden in großen Software Projekten und deren Korrektheit unter verschiedenen Vorraussetzungen, beschäftigte. Als „Trace-Links“ bzw. Pfade bezeichnet man die Verknüpfung einer Anforderung/Aufgabe („Requirement“) mit einem Stück Programmcode (beispielsweise ganze Klassen, oder einzelne Methoden) oder eines Modells mit einer Anforderung, oder zwischen zwei anderen, unterschiedlichen Objekten. Trotz ihrer wichtigen Rolle bei vielen Software-Entwicklungsprozessen und Wartungsarbeiten [1], ist nur wenig über die Auswirkungen des Menschen, auf diesen sehr zeitintensiven Prozess kaum bekannt. Ziel war es, mehr über den „Faktor Mensch“ in diesem Gebiet zu erfahren.

Bei dem Experiment von Univ.-Prof. Dr. Alexander Egyed, M.Sc., Florian Graf und a.Univ.-Prof. Dr. Paul Grünbacher gestaltete sich der Experimentaufbau nicht gerade einfach. So wurden insgesamt 127 Personen aus Wien und Linz herangezogen um an diesem Experiment teilzunehmen. Die 39 davon aus Linz kommenden Studenten mussten das Experiment in einem „kontrollierten Umfeld“ (Labor) erledigen, um jegliche Einflüße von außen zu eliminieren und um genaue Zeitangaben (pro Sitzung waren genau 90 Minuten vorgesehen) zu ermöglichen.

Der zu evaluierende Source-Code für dieses Experiment wurde aus dem OpenSource-Projekt *GanttProject* entnommen, bei welchem es sich um ein Programm zur Projektplanung handelt.
Des weiteren einigte man sich auf 17 Requirements, welche die Grundfunktionen des *GanttProjects* darstellten. Durch zeitaufwendige Vorarbeiten, mittels „Code-Profiling“ und ausgiebigen Tests, wurden aus der riesigen Anzahl von Klassen und Methoden schlussendlich 85 Java-Klassen und 604 Methoden ausgewählt, welche diese 17 Requirements bestmöglichst wiederspiegelten, die es nun zu bearbeiten galt.

Die Studenten in Linz wurden in 2 Gruppen geteilt. Eine war für das Untersuchen der „Trace-Links“ bei den 85 Klassen zuständig, wohingegen der Rest die 604 Methoden evaluierten.

Aufgrund von sehr eindeutigen Ergebnissen in Linz konnten das Experiment in Wien ausserhalb einer kontrollierten Umgebung stattfinden. Ansonsten gab es keine Unterschiede beim Aufbau des Experiments.

Untersucht wurden vor allem die Fragen:

- Erfordert das Wiederfinden von „Traces“/Pfaden eine genaue Kenntnis des Systems?
- Führt mehr Aufwand (zeitlich) zu mehr Erfolg?
- Führt komplexerer Code zu mehr Aufwand und zu niedrigerer Qualität der gefundenen Pfade?
- Benötigen fein-granulare Pfade (in diesem Fall: Requirements auf Methoden und nicht Requirements auf Klassen bezogen) mehr Aufwand und führen sie zu höherer Qualität?
- Benötigen die meisten Experimentsteilnehmer ähnlichen Aufwand?
- Sollte das Auffinden von Pfaden bestenfalls in kurzen, aber dafür regelmäßigen Sitzungen praktiziert werden?

Diesen 6 Hypothesen wurde das Hauptaugenmerk gewidmet und es kam zu teilweise überraschenden Ergebnissen:
So ergab sich zum Beispiel, dass auch ohne genaue Kenntnis des Systems die meisten Teilnehmer sehr ähnliche Ergebnisse erzielten. Erstaunlich vor allem, da ihnen gesagt wurde, sie sollten nur „trace“ oder „no trace“ angeben, wenn sie sich

ganz sicher seien. Sei dies nicht der Fall so stünde noch immer ein „undefined“ zur Verfügung, welches aber nur bei 5% der „Trace“-Stimmen Gebrauch fand.

Als besonders wichtig für die Auswertung der Ergebnisse stellte sich eine genaue Dokumentierung der einzelnen von den Teilnehmern getätigten Aktionen und damit verknüpft eine genaue zeitliche Aufzeichnung selbiger heraus. Dies sollte später auch eine zentrale Rolle in diesem Projekt darstellen.

Zur eigentlichen Durchführung wurde ein kleines von Herrn Univ.-Prof. Dr. Alexander Egyed, M.Sc., entwickeltes „Trace Caputure Tool“ verwendet, welches auch den Grundstein für die hier zu entwickelnde Anwendung gelegt hat.

1.2 Das „Trace Capture Tool“

Das Programm von Herrn Univ.-Prof. Dr. Alexander Egyed, M.Sc., wurde in C# entwickelt. Die Informationen, welcher Benutzer welche Methoden/Klassen zu evaluieren hatte, welche Beziehungen die Methoden zueinander hatten uvm., wurden mittels einer CSV-Datei, pro Benutzer eingelesen.

Wurde also einem Studenten die Kennzahl 100 zugewiesen, musste er diese bei Programmstart eingeben und das Programm las sämtliche Informationen aus, welche in der Datei „100.txt“ gespeichert waren. Das führte vor allem dazu, dass in den verschiedenen Dateien sehr viele redundante Informationen vorhanden waren, welche im Falle eines Fehlers extra geändert werden hätte müssen bzw. sämtliche benutzerbezogene Dateien neu generiert werden hätten werden müssen. Somit war von einer schnellen Wartbarkeit kaum zu sprechen.

Weiters war es notwendig das Programm und die Daten auf jeden Rechner einzeln zu kopieren. Ebenso musste selbiges mit dem Source-Code des zu evaluierenden Projekts geschehen. Dies war auch ein Grund, wieso die oben genannten Experimente quasi nur in einem überwachten Labor durchgeführt werden konnten, wollte man leistungsbezogene Aussagen treffen (wie bei dem Experiment in Linz). Ein anderer Grund war, dass theoretisch jeder seine CSV-Datei verändern hätte können.

Durch die Programmierung mit C# war man ausserdem mehr oder minder an Microsoft Windows gebunden, was bei einer Heimbenützung wohl auch einigen Unmut mit sich gebracht hatte.

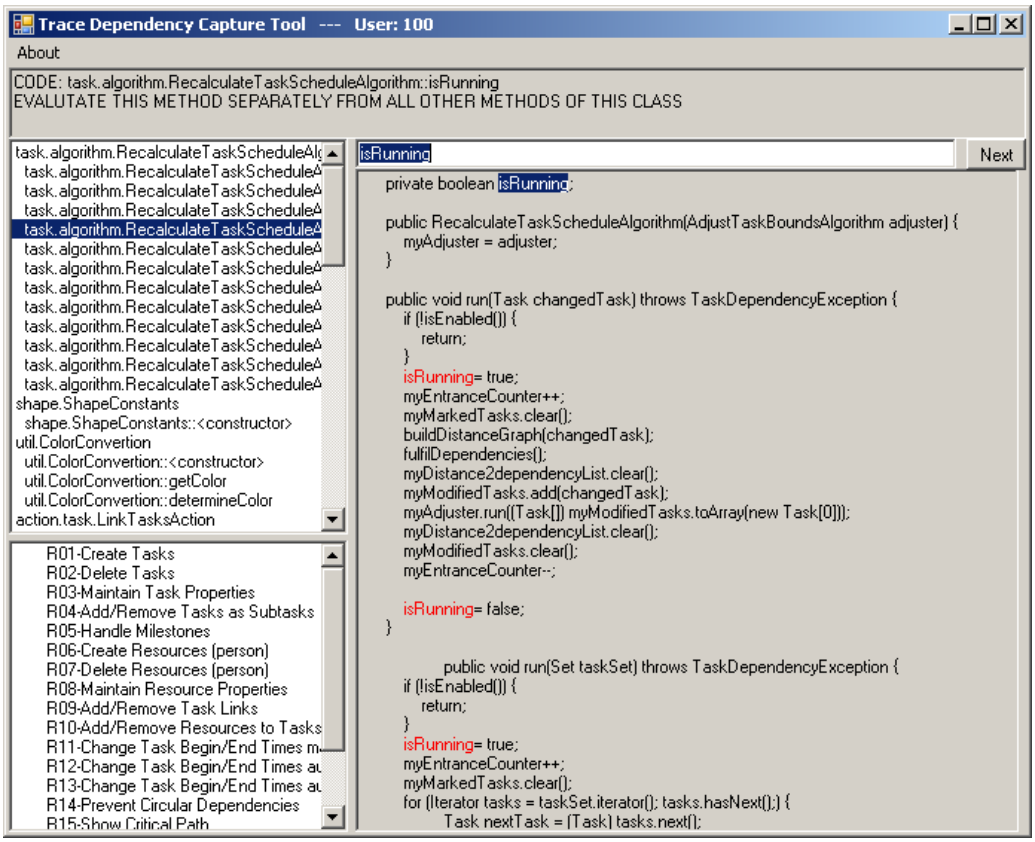


Abbildung 1.1: Bisheriges „Trace Dependency Capture Tool“

1.3 Ablauf

Nachdem sich jeder Student mit seiner Kennzahl angemeldet hatte, musste er Klasse für Klasse, bzw. Methode für Methode, der Reihe nach, auf bestimmte Requirements untersuchen (wie in der Abbildung 1.1 z.B. „R01-Create Tasks“), also ob jene Klasse/Methode diese Anforderung in irgendeiner Weise unterstützt. Anschließend musste der Student für jedes Requirement eine Bewertung abgeben („trace“, “no trace“ oder „potential trace“, wenn man sich nicht sicher war).
Dies musste nun für alle Klassen/Methoden und für die jeweils dazugehörigen Re-

quirements durchgeführt werden, ausser es war ein Zeitlimit vorgegeben nachdem abgebrochen wurde.

Kapitel 2

Grundlagen

2.1 Vorgaben

Ziel des Projekts war es nun eine neue Applikation für die vorher genannten Versuche zu entwickeln.

Die Anwendung sollte dem alten Programm nachempfunden sein bzw. dessen Funktionen enthalten. Jedoch musste sie leichter zu warten und bestenfalls zentral organisiert sein. Weiters stand noch die systemunabhängige Ausführbarkeit im Raum.
Alles in allem ein Projekt prädestiniert für eine webbasierte Implementierung.

Wie schon im C#-Programm musste besonderes Augenmerk auf minutiöses Logging gelegt werden. Die dadurch entstehenden Daten bilden erst die Grundlage für die Auswertung der Experimente. Es war wichtig, dass quasi jeder „Klick“ und jedes betrachtete Element mitdokumentiert werden würde, um eine lückenlose Auflistung der Benutzerinteraktionen zu erhalten.

2.2 Eingesetzte Technologien

Der Entschluss PHP (in der Version 5.xx) zu verwenden und nicht etwa JSP („Java Server Pages”) oder ASP.NET („Active Server Pages .NET“), wurde nach

langen Überlegungen aus mehrerlei Gründen getroffen. Die Kriterien waren vorallem der Aufwand (Einarbeitungszeit, Implementierungsaufwand), Serverunterstützung (schnellstmögliche und unkomplizierte Einrichtung eines Servers, eventuell Betrieb auf Freehost), meine persöhnliche Erfahrung und die Performanz (nicht so wichtig, da das System nicht für mehrere tausend Personen ausgelegt werden sollte).

	JSP/ASP.Net	**PHP**	**Gewichtung**
Aufwand	1	2	2
Serverunterstützung	1	2	3
Performanz	2	1	1
Persöhnliche Erfahrung	1	2	2
Gesamtpunkte:	**9**	**15**	

JavaScript diente vor allem zum Ansprechen mehrere Frames (Arbeitsbereiche) gleichzeitig, was unerlässlich war, sowie auch zum „Highlighten“ von „Strings“ im „Code-Frame“.

Als Unterbau diente des weiteren der Apache WebServer (Version 2.2.9) und eine MySql Datenbank (Version 14.12). Beides OpenSource Produkte, welche quasi auf jedem System/OS eingesetzt werden können.

Entwickelt wurde ausschließlich unter (Ubuntu) Linux und dem Open-Source IDE Eclipse.

Als Test-Betriebssystem kam aber neben Ubuntu Linux auch Microsoft Windows XP zum Einsatz. Es wurden jeweils mehrere Browser (Opera, Firefox, IE7) getestet und etwaige Probleme beseitig.
Gerade im Hinblick auf die Browserkompatibilität lassen Microsoft Produkte immer noch große Wünsche offen. So mussten einige „workarounds“ gefunden werden um einen reibungslosen Ablauf zu garantieren (einzelne Fälle werden im weiteren Verlauf noch genauer beschrieben).

2.3 Aufbau

Der Aufbau sollte ähnlich wie beim Vorgängerprogramm aus drei bis vier Arbeitsbereichen bestehen. Schlussendlich wurden fünf benötigt um eine Art Menü mit Funktionen, die sich je nach Berechtigung ändern können, wie „Abmelden“ und „Passwort ändern“ unterzubringen. Diese Arbeitsbereiche wurden mittels „Frames“ realisiert, welche eine Unterteilung eines (Browser-)Fensters in mehrere Bereiche ermöglicht die sich den Platz je nach Anordnung und Größe teilen. Weiters kann der Benutzer im Nachhinein auch die Größe der einzelnen Frames nach Belieben ändern. Somit kann der gesamte Arbeitsbereich fallweise in der Größe der Bereiche verändert werden, jedoch nicht in der Anordnung.

Abbildung 2.1: Benutzeroberfläche

Wie in der Abbildung 2.1 ersichtlich, ist die linke Hälfte (abgesehen vom „Header-Menü“) zweigeteilt. Hier werden in einem Frame die Knoten samt Unterknoten in einer Baumstruktur dargestellt. Im vorangegangenen Experiment waren dies Klassen und Methoden, grundsätzlich ist aber die Darstellung jeglicher Abhängigkeitsstruktur, die vom Experimentsaufbau vorgegeben wird, möglich.
Im anderen linken Frame werden sämtliche Requirements samt Bewertungsmöglichkeit angezeigt. Mit dem Administrator-Benutzer können die Positionen der beiden Frames wahlweise vertauscht werden.

Auf der rechten Hälfte sind die Positionen fix, wobei sich unten ein großes Frame für die Anzeige etwaiger Inhalte des ausgewählten Knoten befindet. Als Knoten

wird hier ein Objekt bezeichnet, welches mit einem Requirement verknüpft werden kann (also z.B. eine Klasse oder Methode).
Oben rechts werden die „Aufrufstrukturen“/„Verknüpfungsstrukturen“ abgebildet, durch welche navigiert werden kann. Sobald über dieses Frame ein Knoten aufgerufen wird, kann hier mittels eines „Back“-Links wieder einen Schritt Richtung Ausgangsknoten gegangen werden.

Im Header-Frame befindet sich wie oben erwähnt eine Art Menü. Es verfügt über Standardfunktionen wie „Abmelden“ oder „Passwort ändern“ aber auch über ein Eingabefeld für eine Highlighting-Funktion für das große Code-Frame (rechts unten).

2.4 Funktionsumfang

Es wurden zwei Benutzerrollen eingeführt. Neben dem „normalen“ Benutzer kam noch ein Administrator-Benutzer hinzu. Dieser wurde vor allem dazu eingeführt, um Funktionen wie das Zurücksetzen von Passwörtern oder das Erstellen neuer Benutzer, sowie das Einsehen der Log-Aufzeichnungen schnell und ohne zusätzliche Applikationen oder SQL-Abfragen zu ermöglichen.

2.4.1 allgemeine Funktionen

Für einen normalen Benutzer ohne Administratorrechte stellt sich der Funktionsumfang bzw. die Benutzung der Web-Oberfläche wie folgt dar:

Die Startseite besteht aus einer einfachen Benutzername-Passwort-Abfrage, die auf eventuelle Fehler bei der Eingabe hinweist, z.B. leeres Passwort Feld oder falsches Passwort.
Nach erfolgreicher Authentifizierung gelangt man zum eigentlichen Arbeitsbereich. Jedem Benutzer werden jene Knoten, die ihm zugewiesen wurden, in einer Baumstruktur aufgelistet. Weiters sind die zur Verfügung stehenden „Bewertungsmöglichkeiten“ ersichtlich. Mittels eines „Mouseover“-Effekts können genauere Beschreibungen zu den jeweiligen Punkten abgefragt werden.

Im Kopf der Seite befinden sich Links zum sicheren Abmelden sowie zum Ändern des Passwortes.

Sollte die „Session“ des Benutzers abgelaufen sein, also für eine bestimmt Zeit keine Aktionen auf der Seite durchgeführt worden sein, wird er darauf hingewiesen sich erneut einzuloggen. Ansonsten sind keine weiteren Aktionen mehr möglich.

2.4.2 administrative Funktionen

Zusätzlich zu den Funktionen die einem normalen Benutzer zur Verfügung stehen, kann der Administrator-Benutzer noch auf einige weitere Funktionen zurückgreifen.

Die Oberfläche unterscheidet sich nur marginal von der eines normalen Benutzers. Er besitzt jedoch keine Einschränkung im Hinblick auf die Betrachtung der Knoten und Inhalte wie andere Benutzer mit geringerer Berechtigung. Im „Menü“ kommt ein weiterer Verweis - Admin - hinzu, der zu einer Übersicht einiger administrativer Funktionen führt.

Neben dem Anlegen neuer Benutzer lassen sich hier auch die Inhalte der beiden linken Frames vertauschen (im vorangegangenen Experiment wären das z.B. Klassen+Methoden und Requirements gewesen), dies wurde ermöglicht um eine eventuelle neue Versuchssituation zu schaffen, in der geprüft wird, ob durch die andere Aufteilung auch andere Entscheidungen getroffen werden.
Zusätzlich bietet sich die Möglichkeit eine Liste aller Benutzer ausgeben zu lassen, deren Passwörter auf einen Standardwert zurückzusetzen sowie das Betrachten der einzelnen Log-Dateien.

Die Log-Einträge selbst bestehen aus einem „timestamp“, der „page“, die das „Log-Event“ ausgelöst hat und einer etwaigen „action“. Bei der „action“ handelt es sich um Zusatzinformationen über das aufgetretene Ereignis, wie etwa „watched|line:10“, wenn ein bestimmter Knoten betrachtet wurde, oder „added“, “changed“ oder „deleted“ wenn ein Knoten evaluiert wurde. Die wichtigsten Informationen bietet aber bereits die „page“-Spalte:

```
1 low_left.php?file=./code/Cvsread.java
2     &line=81&show=true&trace_id=1&req_id=4&method_id=2&user_id=1
```

Diese Zeichenfolge ist direkt vom Browser entnommen und bietet somit alle Parameter, die auch bei der eigentlichen, internen Verarbeitung nötig sind. Durch die einfache Syntax lassen sich die einzelnen Bausteine leicht anhand der „&“ und der „=“ Zeichen herausfiltern.

Kapitel 3

Implementierung

3.1 Datenstruktur

Als Grundlage für die Datenstruktur wurde auch hier auf die vorangegangenen Entwicklungen zurückgegriffen. Als Vorlage dienten die Eingabedateien des Vorgängerprogrammes, welche anfangs als CSV-Datei und später als XML-Datei zur Verfügung standen. Diese enthielten bereits nahezu alle nötigen Informationen über Zusammenhänge der einzelnen Code-Stücke, sowie Zuweisungen der Code-Stücke an verschiedene Benutzer uvm..

```
0;parentResourceLoadGraphicArea;
  ChartComponentBase    // Vater-Kind-Beziehung
0;requirement;R01-Showstartmenu;
  Launch/Display/Select
  from the startmenu from the taskbar. //Requirement
```

Listing 3.1: Beispiel aus der CSV-Datei eines Benutzers

```
<parent class="ResourceLoadGraphicArea"
    pclass="ChartComponentBase" />
<req id="01" name="Showstartmenu">
    Launch/Display/Select from the startmenu from the taskbar.
</req>
```

Listing 3.2: Beispiel aus der neueren XML-Datei

Weiters wurde bei nahezu jedem Datenobjekt eine „id“ hinzugefügt. Dies dient vor allem zur einfacheren Verarbeitung mit PHP bzw. zur eindeutigen Identifikation jeglicher Elemente in einer Datenbank-Tabelle.

3.1.1 Übersicht

Eine Übersicht über alle in der Datenbank befindlichen Tabellen/Datenobjekte inklusive der Beziehungen untereinander.

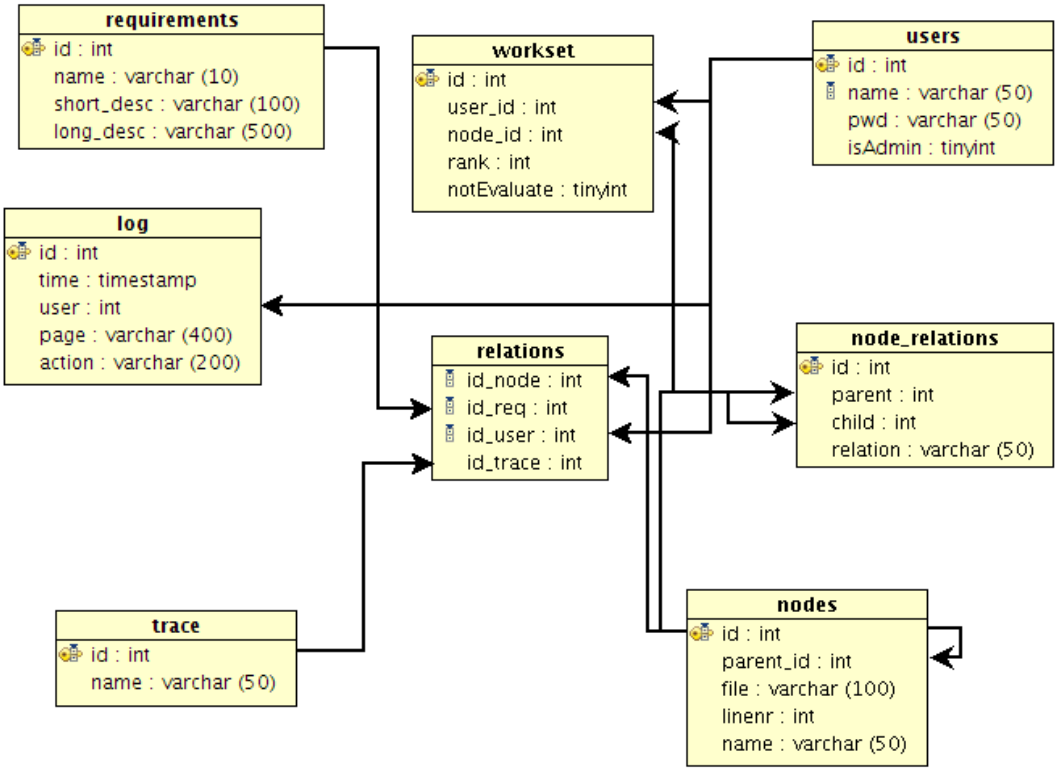


Abbildung 3.1: Überblick über alle Datenobjekte, inklusive Beziehungen

3.1.2 trace

id Automatisch inkrementierter Integer-Wert zur Identifikation innerhalb dieser Tabelle

name Bezeichner, im vorangegangenen Experiment wurden hier Werte: „no trace“, „potential trace“ und „trace“ verwendet.

Im Trace-Datenobjekt werden die späteren Auswahlmöglichkeiten beim Bewerten eines Requirements festgelegt. In der Benutzeroberfläche werden die „name“ Einträge bzw. ihre Anfangsbuchstaben, vor jedem Requirement, angezeigt.
Per „mouseover“-Effekt wird der vollständige Bezeichner angezeigt.

3.1.3 requirements

id siehe oben

name Abkürzung des Requirements, ehemals eine 3 stellige Integer Zahl

short_desc Kurzbeschreibung des Requirements

log_desc ausführliche Beschreibung, wird in der Benutzeroberfläche mittels eines Mouseover-Effekts angezeigt.

Die Requirements stellen die Möglichkeiten dar, die die Experimentsleitung zur Verfügung stellt, welche Funktionen ein Knoten (Klasse/Methode/etc.) haben könnte. Der Benutzer hat nun die Aufgabe, diese Möglichkeiten unter Zuhilfenahme der Trace-Beschreibungen zu bewerten.

3.1.4 log

id siehe oben

time Timestamp, welcher direkt beim Eintragen in die SQL-Datenbank erstellt wird.

user eindeutige „id“ des Benutzers

page Großteil der URL, von der das Logging-Event ausgelöst wurde, samt Parameter

action zusätzliche Information über die getätigte Aktion („watched|line:10“, “added“...)

Diese Logeinträge sind Hauptbestandteil der Auswertung der Experimente. Sie werden pro Benutzer und Aktion (z.B. „Klicken“ eines Knoten) angelegt.

3.1.5 node_relations

id siehe oben

parent „id“ des Vaterknotens

child „id“ des Kindknotens

relation Bezeichnung für die Art der Relation (z.B. „called by“, „called“...)

In dieser Tabelle werden „spezielle“ Abhängigkeiten bzw. Verknüpfungen eingetragen. Diese sind später nicht in der Standardbaumstruktur aufgeführt sondern werden pro Knoten im rechten oberen Frame aufgelistet. Hiermit sollte unter anderem das Navigieren in SourceCode erleichtert werden, da ansonsten Vererbungen verloren gehen würden oder es zu einer zu tiefen und unübersichtlichen Schachtelung in der Baumstruktur käme und etwaige Knoten eventuell nicht mehr gefunden werden könnten.

3.1.6 nodes

id siehe oben

parent_id dieser Wert wird mit -1 initialisiert und später mit der „idëines eventuellen Vaterknotens überschrieben.

file Vollständiger Dateiname samt Pfad (ausgehend vom „Code“-Ordner) der Datei, in der sich der Knoten „befindet“ bzw. wenn die Datei selbst der Knoten ist.

linenr Zeilennummer innerhalb der Datei, in welcher sich der Knoten befindet (ist die Datei an sich der Knoten, wird dieser Wert auf -1 gesetzt)

name Bezeichnung des Knotens (entweder ist dies gleich dem Dateinamen oder z.B. Name einer Methode)

Mit diesem Datenobjekt werden die eigentlichen Knoten beschrieben und mittels „parent_id“ eine Hierachie bzw. Baumstrucktur erzeugt. Bei dieser Hierachie geht es um die Struktur innerhalb einer Datei (bzw. eines Überknotens), also z.B.: Methoden innerhalb einer Klasse oder Überschriften innerhalb eines Textes.

3.1.7 relations

id_node „id“ einer Node

id_req „id“ eines Requirements

id_user „id“ eines Benutzers

id_trace „id“ einer Tracebezeichnung

Verknüpfung der wichtigsten Elemente in dieser Anwendung. In dieser Tabelle wird gespeichert, welcher Benutzer welchen Knoten und welches Requirement wie bewertet hat. Sie gibt den letzten Wert (aktuellen Wert) dieses 4-Tupels an und wird bei einer Änderung überschrieben. Die Tabelle dient vor allem zur korrekten Anzeige in der Benutzeroberfläche. Um eine eventuelle Änderung zu erkennen, was bei der Auswertung sehr wichtig sein kann, müssen die Log-Einträge zuhilfe genommen werden.

3.1.8 users

id siehe oben

name Benutzername, welche beim Anmeldevorgang verwendet werden muss

pwd Passwort des jeweiligen Benutzers welches im PHP-Code MD5-Verschlüsselt verschlüsselt wird.

isAdmin Flag, ob dieser Benutzer Administratorrechte erhalten soll oder nicht.

3.1.9 Workset

id siehe oben

user_id Benutzer, dem dieser Knoten zugewiesen wurde

node_id „id“ des Knotens

rank hier besteht die Möglichkeit einer Reihung der Knoten pro Benutzer, je niedriger die Zahl, desto weiter oben.

notEvaluate dieser Wert bestimmt, ob ein Knoten bearbeitet werden soll oder für diesen Benutzer nur sichtbar ist

Rank wurde wie gesagt eingeführt, um eine Reihung der dargestellten Knoten zu ermöglichen. Dies sollte bei weiteren Experimenten verhindern, dass sich etwaige Gleichheiten in der Bewertung mancher Knoten ergeben, nur dadurch, daß bestimmte Knoten immer am Anfang und andere immer zum Schluss aufscheinen.

3.1.10 control

position name der beiden linken Frames: „up_left“, „low_left“

type „Art“ des Inhaltes (Knoten oder Evaluierungsmöglichkeiten)

Diese Tabelle dient zum Vertauschen der beiden linken Frames, was von einem Administrator initiert werden kann.

3.2 Spezielle Entwicklungsschritte

3.2.1 „Back“-Link

Ein interessantes Feature, welches es zu implementieren galt, war eine Navigationsmöglichkeit durch die Beziehungen der Knoten untereinander. Wurden in vergangenen Projekten ausschließlich „called by“ und „calls“ verwendet, waren die Möglichkeiten bei dieser Anwendung nicht beschränkt.

Abbildung 3.2: „Back“-Link

Um sich gesondert von der „normalen“ Knotenhierachie (wie Methoden in Klassen) ein Bild machen zu können, wurde ein zusätzliches Frame eingeführt, welches sich direkt über dem Frame für die Inhaltsanzeige befindet.
Hier werden neben der Bezeichnung des betrachteten Knoten auch noch oben genannte Beziehungen eingeblendet. Die Verzweigungen führen ihrerseits wieder zu anderen Knoten.

Für das bessere Verständnis und zur leichteren Orientierung musste nun zusätzlich eine Möglichkeit gefunden werden, um ein Zurückschreiten in der Liste der aufgerufenen Knoten möglich zu machen.

Um hier die Funktionsweise und die daraus resultierenden Schwierigkeiten zu verdeutlichen, kurz der Ablauf bei einem Klick auf einen Knoten:
Findet ein solcher statt, werden simultan 3 Frames (Requirements, Inhalt und das Frame welches hier beschrieben wird) gleichzeitig verändert und somit neu vom Server generiert. Dadurch muss penibelst darauf geachtet werden, dass beim Wechseln zu einem anderen Knoten alle 3 Frames konsistent gehalten werden.

Eine anfängliche Idee war ein simpler Aufruf der JavaScript history.back()-Funktion. Mit deren Hilfe konnte man einen Schritt (oder mit einem optionalen Parameter eine bestimmte Anzahl von Schritten) in einer „History“ zurückgehen und somit zur vorigen Seite gelangen. Durch die Verwendung von Frames wurde diese Aufgabe aber nicht ganz so trivial lösbar, da dieser Rückschritt immer nur ein Frame betraf. So wurden bei drei geänderten Frames (Requirements, Inhalt, Beziehungen) auch drei Einträge in die History gemacht. Dank des erwähnten Parameters wäre auch dies kein Problem gewesen (history.back(3)), doch wurden auch bei jedem weiteren Linkaufruf (z.B. Anklicken einer Trace im „Requirementsframe“) ein weiterer History-Eintrag getätigt. Somit würde anschließend bei einem Rückschritt um nur 3 Stufen ein inkonsistenter Zustand entstehen.
Beispielsweise, würden im Requirementsframe die Daten für die Methode „readMyFiles“ angezeigt werden, ebenso im „Beziehungsframe“, während aber im Inhaltsframe noch immer die vorher geöffnete Methode „writeMyFiles“ zu sehen wäre.
Dies würde schnell zu Falschinterpretationen der Anzeige führen und somit das Ergebnis der kommenden Experimente enorm beeinflussen.

Nachdem sich die erste Version als unzureichend herausstellte, wurde ein anderer Ansatz gewählt. Eine eigene Implementierung eines Kellerspeichers („stacks“,

„first in, last out“) mit Hilfe von PHP sollte schlussendlich zu einer Lösung führen. Im Grunde sollte die Funktion dieselbe sein, wie die der JavaScript-history.back()-Funktion, doch an die jeweiligen Verhältnisse angepasst.

Hauptziel war natürlich, dass weiterhin und trotz Implementierung der neuen Funktionalität die Konsistenz über alle Frames hinweg bewahrt werden sollte. Weiters sollte der „Back-Link“ nur erscheinen, wenn wirklich eine Navigation im „Knoten-Beziehungsframe“ stattfand.
Aufrufe von „ausserhalb“, also vom „Hierachieframe“ herrührend, sollten den Kellerspeicher wieder leeren, da somit ein neuer Pfad beschritten werden würde. Um einen solchen Aufruf aus einem anderen Frame zu identifizieren war die logischste Lösung das Verwenden des Http-Referers. Mittels der vordefinierten, globalen Variable $_SERVER["HTTP_REFERER"] kann auf die gesamte URL des letzten Links, der auf diese Seite verweist, zugegriffen werden. Zusätzlich kam noch die PHP-Funktion substring() zum Einsatz, um die Bezeichnung des „Ausgangsframes“ herauszufiltern. Diese Methode funktionierte auch bei den meisten Browsern sehr gut. Nur der Microsoft Internet Explorer 7 verweigerte strickt die Zusammenarbeit. Anstatt der Bezeichnung des Frames, aus welchem der Aufruf kam, lieferte die Variable beim IE7 die Bezeichnung der Datei, in welcher das gesamte „Frameset“ definiert wurde, was das ganze Vorhaben ad absurdum führte.

Als endgültige Lösung bewährte sich dann ein zusätzlicher Parameter, der jedem Link im Frame angehängt wurde und auf diesem Weg sicherstellte, dass nur bei der Navigation innerhalb des rechten oberen Frames ein „Back-Link“ angezeigt wurde.

Die Informationen, die zur korrekten Ausführung dieser Abfragen von Nöten waren und somit auch zur Befüllung des Kellerspeichers dienten, wurden mittels Parametern an das Frame übergeben. Der Stack selber wurde in der „Session“ global und pro Benutzer gespeichert. Als Session wird eine Verbindung zwischen Client (Benutzer) und Server (in diesem Fall dieses PHP-Programm, bzw. der darunterliegende Server) bezeichnet, die beim Login aufgebaut wird und nach einer gewissen inaktiven Zeit wieder endet, oder explizit durch das Ausloggen abgebaut wird. Während diese Session besteht, können Informationen darüber ausgetauscht bzw. für die Dauer der Session gespeichert werden. Nach Beendigung der Session werden diese Informationen wieder verworfen.

```
1 if ($_GET['pop'] == 1)
```

```
2 {
3   array_pop($_SESSION['stack']);
4 }
5 $count = count($_SESSION['stack']);
6 if (!$_SESSION['stack'])
7 {
8     $stack = array (null);
9     $_SESSION['stack'] = $stack;
10 }
11 shownodeRelations($_GET['node'], $_GET['target'], true, $stack);
12 if ($_GET['q'] != 1 && $count > 1 &&
13   $_SESSION['stack'][($count-1)][node] != $_GET['node'])
14 {
15     session_unregister('stack');
16     $stack = array (null);
17     $_SESSION['stack'] = $stack;
18     array_push($_SESSION['stack'], array ('node'=>$_GET['node'],
19   'target'=>$_GET['target'], 'file'=>$_GET['file'],
20   'line'=>$_GET['line']));
21 }
22 $count = count($_SESSION['stack']);
23 $ar1 = $_SESSION['stack'][($count-2)];
24 if ($count > 2)
25 {
26   echo "<a href='..'>Back</a>";
27 }
```

Listing 3.3: Implementierung des Back-Links

3.2.2 Multible Frames ändern

Mit reinem HTML bzw. PHP ist leider keine Methode bekannt, wie man mehrere Frames auf einmal ändern könnte. Mit herkömmlichen HTML-Links (xyz kann pro „Klick"nur ein Frame angesprochen werden. Deshalb musste auf JavaScript zurückgegriffen werden. Mit dessen Hilfe können mittels eines „normalen" Links noch zusätzliche Anweisungen ausgeführt werden.

```
1 <a href="x.php" target="right"
2   onClick="FrameAendern('y.php','left')">Link</a>
```

Als Ausgangspunkt diente ein kleines Skript mit dem maximal zwei Frames geändert werden konnten. Dieses sohingehend erweitert, dass die Anzahl der maximalen Frames aufgehoben wurde.

```
1 function FrameAendern(){
2   for (var i = 0; i < FrameAendern.arguments.length; i = i + 2) {
3     parent[FrameAendern.arguments[i + 1]].location.href
4           = FrameAendern.arguments[i];
5   }
6 }
```

Listing 3.4: FrameAendern()

Als Parameter für die Funktion FrameAendern() sind pro Frame immer zwei Angaben zu machen. Zum einen der eigentliche Link, welcher sich nicht vom href-Teil eines normalen HTML-Links unterscheidet. Zum anderen ist das Ziel des Links anzugeben, also die Bezeichnung des Frames, welches im Frameset definiert wurde.

3.2.3 Eingabedatei Parser

Neben der eigentlichen Anwendung wurden noch einige kleinere „scripts“ benötigt um den reibungslosen Ablauf zu garantieren. Allem voran war die Entwicklung eines Parsers der Eingabedateien wichtig. Anfangs erfolgte die Eingabe einiger Testeinträge noch manuell, direkt über die „phpmyadmin“-Oberfläche in die MySQL-Datenbank. Für größere Datenmengen wäre dies aber unnötig zeitaufwendig und unpraktikabel, da diese Oberfläche eher zur Organisation der Datenbank gedacht ist, als zur Verarbeitung komplexer Daten. Da neben diesem Projekt ein weiteres lief, welches die Neustrukturierung und Generierung der Eingabedaten, auf XML-Basis, zum Ziel hatte, konnte man auf die wesentlich schöner und einfacher zu parsende XML-Datei zurückgreifen.

Mittels der in PHP5 eingeführten Erweiterung „SimpleXml“ war es kein Problem jedwede Art von XML-Dateien zu parsen. Die Syntax erlaubte es einerseits direkt auf ein Element zuzugreifen als auch einen ganzen Teilbaum abzufragen und diesen in einer Schleife zu durchwandern.

```
1 <?xml version="1.0"?>
2 <tdc>
```

```
3   <req id="01" name="Create Tasks">Create a new task which typically
4   has a start date and an end date.</req>
5 </tdc>
```

Listing 3.5: Auszug aus io2.xml

```
1 $xml = simplexml_load_file('io2.xml');
2 function getReq($xml){
3   $path = "//req";
4   $reqs = $xml->xpath($path);
5   foreach ($reqs as $req)
6   {
7     $insert = array ('name'=>$req['id'], 'short_desc'=>$req['name'],
8       'long_desc'=>htmlspecialchars($req));
9     DB::insert('requirements', $insert);
10  }
11 }
```

Listing 3.6: Auszug aus dem Eingabedatei-Parser

Mittels XPath-Ausdrücken ist es möglich durch den XML-DOM zu navigieren. Im obigen Beispiel werden mit $path = "//req";$reqs = $xml->xpath($path); alle „req“ Elemente ausgewählt. Auf die Attribute des jeweiligen Elements kann mit Hilfe von „SimpleXml“ wie auf ein Array zugegriffen werden $req['id']. Anschließend werden die extrahierten Informationen in ein Array gespeichert ($insert) und über die Hilfsmethode DB::insert in die Datenbank eingefügt.

Da bei den Eingabedaten die „parent-child“-Beziehung nicht über die „id“s der jeweiligen Knoten angegeben war, sondern mittels der „Namen“ und zusätzlich noch in einem extra XML-Element (also nicht im Rahmen der „Code“-Tags), musste noch ein weiterer Schritt, zusätzlich zum reinen Einfügen der Daten, gemacht werden.
Es war nötig, nach dem Einfügen aller „Code“-Elemente in die Datenbank, ein „update“ über sämtliche „parent_id“s der „method“-Tabelle laufen zu lassen. Zuvor wurden mit Hilfe der „namen“ noch die „id“s der einzelnen Elemente ermittelt.

Dies ist nur eine Struktur mit der sich die Datenbank befüllen lässt. Weiters wurden DTD-Dateien („Document Type Definition“) und XML-Schema-Dateien erstellt, die als Vorlage für eine neu erstellte XML-Datei dienen und welche eine leicht vereinfachte Version des oben vorgestellten Parsers parsen könnte.

Wie in diesem Beispiel kann die gesamte Datenbank befüllt werden.

3.2.4 „Mehr Generizität“

Ein weiteres Ziel das gesetzt war, doch schlussendlich nicht rechtzeitig fertiggestellt werden konnte, war mehr Generizität in die Datenstruktur und somit in die gesamte Anwendung zu bringen.
Unabhängig vom Inhalt sollte jedes Frame (bzw. zumindest die beiden linken) dasselbe anzeigen können, bzw. mit den selben Daten arbeiten können. Durch die Vielzahl an Möglichkeiten, die bei der paarweisen Verknüpfung von z.B. Code-Stücken, Modellen, Test-Cases, Requirements mit jeweils unterschiedlichen Datenstrukturen entstanden, war es in der gewünschten Zeit leider nicht mehr möglich ein brauchbares Resultat zu erzielen.

3.2.4.1 Variante 1

Der erste Ansatz brachte in Wirklichkeit keine höhere Generizität, sondern eher eine Erhöhung der Möglichkeiten, die 3 zur Verknüpfung dienenden Datenobjekte, Nodes, Requirements und Traces, anzuordnen. Es sollte also möglich gemacht werden, diese 3 Objekte nach Belieben anzuordnen bzw. auf die zwei linken Frames zu verteilen. Der Maximalfall pro Frame hätte dann wie folgt aussehen können: Jede Node hat eine unbeschränkte Anzahl von Unter-Nodes. Jeder Knoten (Node) hat die Bewertungsmöglichkeiten ([t][n], „trace“ „no trace“) direkt nebst den vordefinierten Requirements. Diese Anordnung würde das zweite linke Frame überflüssig machen.

```
1  Node1
2    [t][n]Requirement1
3    [t][n]Requirement2
4    ..
5     Node1.1
6       [t][n]Requirement1
7       [t][n]Requirement2
8       ..
9  Node2
10   [t][n]Requirement1
11   [t][n]Requirement2
```

```
12      . .
```

Listing 3.7: Maximalfall pro Frame

Durch die beschränkte Anzahl von Möglichkeiten war es eine in der Theorie sehr einfache, aber codeaufwendige Lösung. Die insgesamt 7 verschiedenen Möglichkeiten bestanden aus: Node, Requirement, Trace, Node+Requirement, Node+Trace, Requirement+Trace, Node+Requirement+Trace. Diese wurden wiederum in einzelne Funktionen aufgeteilt:

- showNodes: für N(odes), N+T(races), N+R(equirements), N+R+T
- showRequirements: für R, R+T
- showTraces: für T

Jeder Funktion wurde mittels Parametern mitgeteilt, was sie von ihren Möglichkeiten anzeigen sollte. Intern wurde anschließend ein „IF"-Baum durchwandert, der die einzelnen Kombinationen darstellte. Die reine Anzeige der einzelnen Objekte war kein großes Problem, dafür jedoch die Erstellung passender Links umso mehr.

Für jede Kombination mussten unterschiedlichste Verweise erstellt werden, zusätzlich musste quasi der Link selber noch wissen, was im anderen Frame angezeigt wird bzw. nach der Aktivierung des Links angezeigt werden sollte.
Weiters war noch immer die Frage offen, welche Kombinationen überhaupt sinnvoll wären.

Aufgrund solcher und ähnlicher Komplikationen wurde diese Variante zwar teilweise implementiert, doch schließlich auch aus zeittechnischen Gründen auf Eis gelegt.

3.2.4.2 Variante 2

Einer weiteren Annäherung an ein komplett generisches System lag die Idee zugrunde, die Datenbank-Struktur für Nodes und Requirements anzugleichen, um beide ident verarbeiten zu können. Diese Lösung wäre in wenigen Zeilen zu implementieren gewesen, doch wurde ausser Acht gelassen, dass ein wichtiger Teil

zur Verknüpfung fehlte: die Traces. Eine Beziehung zwischen Nodes und Requirements war wenig sinnvoll bzw. gar nicht möglich, da die Verknüpfungsbasis (nämlich die Trace-Bewertungsmöglichkeiten) fehlte.
Um möglichst wenig Zeit bei der Entwicklung des eigentlichen Projekts zu verlieren, wurde diese Variante nicht implementiert, obwohl sie wahrscheinlich die bessere Möglichkeit dargestellt hätte, etwas mehr Generizität in die Applikation zu bringen.

3.3 Allgemeine Probleme

Neben den bereits beschriebenen Problemen traten natürlich auch noch ganz alltägliche Fragestellungen auf, wenn es darum ging, ein Programm, das in C#, Java oder einer ähnlichen Sprache programmiert wurde und nicht an einen Browser gebunden ist, in eine Webapplikation zu überführen.

Lange wurde überlegt, wie man die Funktionalität der im „alten“ Programm mit Rechtsklick erreichbaren Elemente am besten einbaut ohne die Bedienung erheblich zu erschweren. Manches konnte direkt ohne Umschweife angezeigt werden (Bewertungsmöglichkeiten bei den Requirements zum Beispiel), andere Punkte wie etwa die Aufrufsstrukturen („called“,“called by“) wurden in ein eigenes Frame (mit Zusatzfunktion, siehe 3.2.1) ausgelagert.

Weiters, wie schon mehrfach erwähnt, gab es mehrfach Schwierigkeiten bei der so genannten Browserkompatibilität. Kaum ein Browser hält sich an bestehende Standards (und aus diesem Grund wohl auch sehr wenige Entwickler), was es schwierig macht, eine Applikation zu entwickeln, welche in allen (bzw. den populärsten Browser, in diesem Fall wurden IE7, Firefox und Opera gewählt) Browsern gleich aussieht und sich auch gleich verhält. Deshalb sind auch einige geplante Gui „Verschönerungen“ auf der Strecke geblieben, da der Aufwand im Verhältnis zum Nutzen einfach in keiner brauchbaren Relation stand. So musste die „Benutzeroberfläche“ so puristisch wie möglich belassen werden. Hier gäbe es sicher noch einen großen Nachholbedarf.

Doch nicht nur beim Aussehen musste aufgepasst werden, wie sich jeder einzelne Browser verhält. Viel schwerwiegender warn die teils verschienden Interpretationen von JavaScript-Befehlen oder globalen PHP-Variablen (siehe 3.2.1). Teils

mussten dadurch große „Umwege“ in Kauf genommen werden, um das gewünschte Ergebnis zu erreichen.

Kapitel 4

Zukunftsausblick

Eine Weiterentwicklung in vielerlei Richtungen ist möglich. Sei es nun andere Technologien zu verwenden, z.B. in Richtung JSP, oder die Performanz zu steigern. Gerade in der Codeoptimierung sind sicherlich noch einige Möglichkeiten gegeben. Auch das Verwenden des neuen objektorientierten Ansatzes von PHP5 würde sicher zur Leserlichkeit des Codes beitragen.

Weiters ist die Erhöhung der Generizität ein wichtiges Thema, welches es zu bearbeiten gilt und in kommenden Implementierungen sicher eine wichtige Rolle spielen wird. Wie schon besprochen wurden einige Ansätze angedacht und auch teilweise realisiert, doch ohne hinreichende Erfolge. Eine neue Herangehensweise und mehr Zeit wären von Nöten, um hier ein befriedigendes Ergebnis zu erzielen.

Im Hinblick auf zukünftige Experimenten könnten auch die Inhalte ausgetauscht werden (was von der Applikation bereits unterstützt wird). Neben JavaCode kann quasi jeglicher Dateninhalt angezeigt werden, wobei für über 80 Sprachen, dank *GeSHi*[2] (PHP-Script), ein Syntaxhighlighting zur Verfügung steht. Zusätzlich könnten auch Bilder oder PDF-Dokumente evaluiert werden. Eine passende Fragestellung (Requirements und Traces) ist natürlich vorauszusetzen.

Literaturverzeichnis

[1] Paul Grünbacher Alexander Egyed, Florian Graf. Recovering Trace Links between Requirements and Code: Understanding the Human in the Loop. Technical report, Institute for Systems Engineering and Automation, Johannes Kepler University Linz, Austria.

[2] GeSHi. GeSHi - Generic Syntax Highlighter http://qbnz.com/highlighter.

[3] The PHP Group. PHP-Dokumentation http://www.php.net.