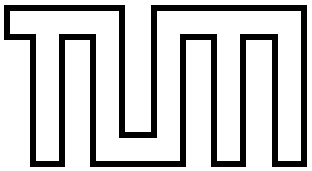


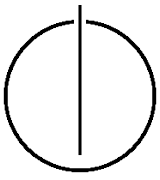
FAKULTÄT FÜR INFORMATIK

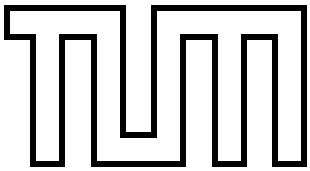
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Visualisierung der Evolution von strukturierten Wikiseiten mit Typen, Attributen und Integritätsbedingungen - Analyse, Design und prototypische Implementierung

Manuel Tremmel





FAKULTÄT FÜR INFORMATIK

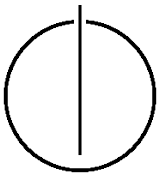
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Visualisierung der Evolution von strukturierten Wikiseiten mit Typen, Attributen und Integritätsbedingungen - Analyse, Design und prototypische Implementierung

Visualizing the Evolution of Structured Wiki Pages with Types, Attributes, and Constraints - Analysis, Design und prototypical Implementation

Bearbeiter:	Manuel Tremmel
Aufgabensteller:	Prof. Dr. Florian Matthes
Betreuer:	Christian Neubert
Datum:	14. Mai 2012



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Garching, den 14. Mai 2012 Manuel Tremmel

Abstract

Hybride Wikis stellen einen Mittelweg zwischen traditionellen Wikis und semantischen Wikis dar. Sie ermöglichen die inkrementelle Strukturierung von unstrukturierten Informationen mit leichtgewichtigen, flexibel kombinierbare Strukturierungskonzepten (Attribute, Typen und Integritätsbedingungen) sowie Mechanismen (z.B. Attributvorschläge). In dieser Arbeit soll die strukturelle Evolution von hybriden Wikis untersucht und visualisiert werden.
Zu diesem Zweck werden verschiedene Anwendungsdaten auf ihre Eignung als Datenquelle für die Analyse überprüft, so dass verschiedene Forschungsfragen bezüglich der Nutzung von Strukturierungstechniken beantwortet werden können. Außerdem wird untersucht wie man dem Nutzer die Möglichkeit geben kann, eigene Abfragen an das System zu stellen. Daraufhin wird ein leichtgewichtiges Tricia-Plugin entwickelt, das Anwendungsdaten produktiver Hybrid-Wiki-Installationen auswertet und so eine Analyse der strukturellen Evolution ermöglicht. Die Evolution wird dabei dann von verschiedenen Dashboards visuell aufbereitet; zudem erhalten Benutzer die Möglichkeit eigene Abfragen zu erstellen.
Anhand konkreter Anwendungsdaten soll mit Hilfe dieses Plugins eine programmatische Analyse durchgeführt, ausgewertet und interpretiert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

Bei dem vom Lehrstuhl Software Engineering for Business Information Systems entwickelten Hybrid Wiki handelt es sich um ein generisches Tool, das Strukturierungsmechanismen und -konzepte zur Verfügung stellt, die es auch Nicht-Experten ermöglichen sollen semantisches Wissen auf intuitive Art und Weise zu speichern und mit Anderen zu teilen. Dieses Tool konnte mit einer überschaubaren und relativ geringen Anzahl von Konzepten – nämlich Attributen, Typen, Attributdefinitionen und Typdefinitionen – einen Mittelweg zwischen rein textbasierten, nicht-semantischen Wikisystemen (wie Wikipedia[1]) und semantischen Wikis gehen, wobei für letztere aufgrund komplexer Strukturierungskonzepte ein Schema-Designer nötig sein kann [MNS11].
Aus wissenschaftlicher Sicht ist es sehr interessant herauszufinden, wie die Strukutierungskonzepte und -mechanismen genutzt werden. Mit Hilfe dieser Erkenntnisse kann dann beispielsweise die Benutzerführung so optimiert werden, dass die von Hybrid Wiki angebotenen Funktionen optimal genutzt werden.
In dieser Bachelorarbeit werde ich ein Tricia-Plugin entwickeln, das Fragen zur Nutzung der Strukutierungskonzepte und -mechanismen beantworten hilft. Neben vordefinierten Fragen ist es insbesondere interessant, dem Benutzer des Plugins die Möglichkeit zu geben, auch eigene Abfragen zu tätigen. Wie man eine solche Möglichkeit implementieren kann, soll auch im Rahmen dieser Bachelorarbeit beschrieben werden. Des Weiteren soll auch die Frage beantwortet werden, wie die aus den Abfragen gewonnenen Daten visualisiert werden können.

1.2 Aufbau der Arbeit

Zunächst werde ich in Kapitel 2 meine Hintergrundrecherchen beschreiben, die bei Analyse, Design und Implementierung dieser Bachelorarbeit oder bei der Benennung meiner Plugin-Funktionalität hilfreich waren.
In Kapitel 3 sollen Tricia und Hybrid Wikis vorgestellt werden, wobei der Fokus auf der Funktionalität liegen wird, die für die Bearbeitung der Bachelorarbeit – insbesondere der prototypischen Implementierung eines Tricia-Plugins – relevant war.
Kapitel 4 widmet sich der Analyse, um Anforderungen an das zu entwickelnde Plugin zu identifizieren.
Das Design des Plugins wird in Kapitel 5 diskutiert, wobei auch versucht werden soll, verschiedene Lösungsansätze zu bewerten.
In Kapitel 6 wird die Implementierung, Laufzeit und das Testverfahren beschrieben.

[1] http://de.wikipedia.org/ (aufgerufen am 24.04.2012)

Das Kapitel 7 beschreibt die Integration, Verwendung und Bedienung des Plugins, insbesondere wie Benutzer eigene Abfragen in Wikiseiten einbetten können.
In Kapiel 8 werden die mithilfe des Plugins gewonnenen Erkenntnisse exemplarisch für die öffentliche Website des sebis Lehrstuhls dargestellt.
Ein Fazit dieser Bachelorarbeit ist in Kapitel 9 zu finden.

Hinweis zur Notation
In dieser Schriftart werden – in der Regel englische – Fachbegriffe dargestellt.
Beispiel: type tags

2 Grundlagenrecherchen

Um die Grundlagen für die Bearbeitung dieser Bachelorarbeit zu schaffen, werden nun Bereiche beschrieben, die mit der erforderlichen Nutzungsanalyse thematisch verwandt sind.

2.1 XML und XPath für Data Mining und Knowledge Discovery

XML und XPath als Grundlage für Informationsextraktion zu verwenden, schafft – nach den Autoren des scientific papers [CK05] „Trends in Data Mining and Knowledge Discovery“– ein mächtiges Werkzeug für Data Mining und Knowledge Discovery (DMKD). In Zeiten, in denen große Mengen an Wissen in Datenbanken gespeichert wird, stellen die Schlüsse, die man daraus durch DMKD gewinnen kann, für Unternehmen meiner Meinung nach einen großen Wettbewerbsvorteil dar. Als Vorteile von XML werden

1. die Standardisierung durch das W3C-Konsortiums,
2. die Plattformunabhängigkeit des XML-Standards,
3. zahlreiche Tools, die XML-Datenmanipulation ermöglichen
4. sowie die große Verwendung, die XML bereits in IT Projekten findet,

genannt [CK05, vgl. Abschnitt 1.1.1, S. 2, Z. 18-38].
Als spezielle Vorteile von XML für DMKD wird angeführt, dass eine Speicherung der Daten von verschiedenen Datenbanksystemen im XML Standard verschiedene Tools für verschiedene Datenbanksyteme überflüssig machen könnte und eine Analyse und Kommunikation zwischen verschiedenen Datenbanksytemen erleichtert [CK05, vgl. Conclusion, S. 21, Z. 4-15]. Dieser Vorteil würde im Fall eines DMKD-Prozesses auch für Tricia gelten, das mit zahlreichen Datenbanksystemen zusammenarbeiten kann. Zu den Data Mining Methoden gehören unter anderem auch machine learning, neuronale Netze und clustering [CK05, vgl. S. 7, Z. 14-17]. Die Kommunikation via XML führt nach den Autoren zu der Möglichkeit, Frameworks für DMKD zu entwickeln und in den verschiedenen Schritten der DMKD verschiedene Tools anzuwenden.
Ähnliches ist auch mit dem Plugin möglich, das im Rahmen dieser Bachelorarbeit entstanden ist. So wird etwa aus der Datenbank ein XML-Dokument erstellt, auf dem man eigene XPath-Abfragen erstellen kann. Die Ergebnisse der XPath-Abfragen können dann an die Visualisierungstools verschiedener Anbieter weitergegeben werden.

2.2 XML Datenbanken

XML kann Daten oftmals besser abbilden als relationale Datenbanken, da XML Baum-Strukturen speichern kann, während das in relationalen Datenbanken nur über Umwege

funktioniert. Deshalb sieht Edmond Andrew N. in seinem paper „On data mining Tree structured data represented in XML“ einen Trend von den relationalen Datenbanken hin zu XML-nativen Datenbanken und objektorientierte Datenbanken vor allem für spezielle und komplexere Anwendungen [Edm03, S. 1, Z. 10-S. 2, Z. 6]. Vor allem zur Repräsentation von Baum- und baumähnlichen Datenstrukturen sind XML-Datenbanken besser prädestiniert. Es gibt XML-native und XML-enabled Datenbanken:

- XML-Native Datenbanken unterscheiden sich von relationalen Datenbanken darin, dass statt der Tupel XML-Dokumente als fundamentale Speichereinheit verwendet werden. Manche dieser Datenbanken werden über ein relationales, objektorientiertes Modell definiert, andere sind schemaunabhängig [CK05, vgl. S. 12, Z. 17-31].
- XML-enabled Datenbanken können neben den üblichen Tupeln auch XML-Dokumente speichern. Allerdings wurde die XML-Datei anfangs, also im Jahr 2005, oftmals verlustbehaftet gespeichert, da etwa processing instructions und die Reihenfolge der tags verloren gehen [CK05, vgl. S. 12, Z. 32-46].

Bei Tricia wurde die Designentscheidung getroffen, Daten nicht rein-relational zu speichern. Stattdessen werden Daten im nicht-relationalen Format JSON innerhalb der Datenbank abgelegt. Das zeigt, dass die rein relationale Vorgehensweise ineffizient sein kann bzw. zu viel overhead bedeutet würde.

2.3 Terminologie

In dieser Bachelorarbeit wird der Begriff snapshot verwendet, um die einzelnen Versionen (also Abbilder) der WikiPage zu bezeichnen. Der Begriff snapshot stammt aus der Fotografie und bedeutet Schnappschuss [LR07].

Snapshots in Data Backups Der Begriff snapshot für Abbilder der Wikiseiten spielt auch auf den Begriff snapshot im Bezug auf Backup-Systeme an. So gibt es verschiedene Technologien, um bei dem länger andauernden Prozess des „Abfotografierens“ des Festplatteninhalts zu vermeiden, dass zwischenzeitliche Änderungen bis zum Abschluss der Backup-Erstellung mit in dem snapshot aufgenommen werden. Dieses Phänomen ähnelt der Belichtungsdauer beim Fotoapparat, die - wenn zu lange eingestellt - keine tatsächliche Momentaufnahme mehr liefert, sondern ein verschwommenes Bild.
Unter einem vollständigen Backup versteht man eine 1:1 Kopie des Speichermediums zu einem bestimmten Zeitpunkt. Wenn regelmäßig Backups erstellt werden, kann die für die Backups benötigte Kapazität schnell sehr groß werden; außerdem sind full backups sehr zeitaufwändig. Aus diesem Grund können basierend auf einem vollständigen Backup eine Reihe von inkrementellen Backups erstellt werden. Inkrementelle Backups erhalten lediglich die Änderungen seit dem letzten Backup. Deshalb ist die Größe eines inkrementellen Backups relativ gering. Gewöhnlich werden full backups und incremental backups kombiniert, indem gelegentlich vollständige und häufig inkrementelle Backups erstellt werden [CVK98].
In Tricia stehen - analog zu Dateisystem-Backups - auch ein full backup zur Verfügung, nämlich der aktuelle Zustand der WikiPage, der Attribut- und type tag-Definitionen. Alle

vorigen Abbilder können regeneriert werden, indem die inkrementell gespeicherten ChangeSets wiederhergestellt werden. ChangeSets sind somit inkrementelle Backups (in der Datei-System-Terminologie).

3 Tricia

Bei der open source-Software Tricia handelt es sich um eine Kollaborationsplattform, also eine Software die den Informationsaustausch fördern soll. Tricia stellt zu diesem Zweck unter anderem ein Wikisystem mit Wikiseiten zur Verfügung, die mittels WYSIWYG-Editor bearbeitet werden können und in verschiedenen Wikis abgelegt werden können[1]. Auf diesen Wikiseiten baut das Plugin HybridWiki auf, indem es Strukturierungstechniken zur Verfügung stellt[Neu12].

3.1 Model-View-Controller

Tricia verwendet das Model-View-Controller Design Pattern. Im Folgenden sollen Datenschicht (Model), Präsentation (View) und Geschäftslogik (Controller) in Tricia vorgestellt werden. Das Augenmerk liegt dabei auf den Aspekten, die auch bei der Implementierung des Plugins benötigt wurden.

Model und ORM

Um für Datenpersistenz zu sorgen werden Java-Klassen, die Subklassen von PersistentEntity sind, in einer Datenbank gespeichert. Tricia kann mit vielen verschiedenen Datenbanksystemen arbeiten; so können derzeit z. B. Tricia-Daten in den Datenbanksystemen (DB2, HSQLDB, MongoDB, MSSQL, MySQL und Oracle) gespeichert werden. Für das Mapping der Java-Objekte in die Datenbank wird ein Tricia-eigenes Object Relational Mapping (ORM) verwendet. [Mat12b].

DMKD für Tricia Um im Rahmen eines DMKD-Prozesses Schlüsse aus den Tricia-Daten zu gewinnen, müssten aufgrund der Vielzahl von unterstützten Datenbanksystem je nach verwendetem Datenbanksystem ggf. verschiedene Tools eingesetzt werden. Eine besondere Problematik für Datenextraktion aus den aktuellen Tricia-Datenbanken ist, dass die JSON-kodierten Tabellenfelder per SQL nicht mit nativen Methoden des Datenbanksystems (z. B. MySQL) ausgelesen werden können. Das bedeutet, dass keine MySQL-eigenen Funktionen zur Verfügung stehen, um JSON-kodierte TEXT bzw. VARCHAR-Felder auszulesen. So kann etwa nicht mit Hilfe von SELECT-Operationen nach JSON-kodierten Daten sortiert werden noch danach in der WHERE-clause gefiltert werden. Stattdessen werden die Daten komplett ausgelesen und im Anschluss von Java-Methoden gefiltert bzw. sortiert. Nachteilig ist in diesem Fall aber der overhead, der dann in gewissen Situationen entstehen kann.
Beispielsweise kann etwa für JSON-kodierte Inhalte kein Index angelegt werden; bei sehr

[1] vergleiche `http://www.infoasset.de/wikis/infoasset/instant-productivity-gains` (Aufgerufen am 24.04.2012)

großen Datensätzen könnte deshalb die Performance leiden, wenn nach Daten, die im JSON kodiert sind, gefiltert oder sortiert werden soll. Oder: Möchte man beispielweise sämtliche Änderungen, die den Volltext betreffen, aus dem ChangeSet auslesen, so kann per SQL keine Filterung vorgenommen werden, um andere Arten von Änderungen nicht in der Auswahl miteinzubeziehen.
Der Grund für die Entscheidung für die JSON-Encodierung in den SQL-Datenbanken im Rahmen des Designprozesses von Tricia ist vermutlich die größere Flexibilität, mit der Daten gespeichert werden können. So können z. B. „on the fly" Tree-förmige, neue Datenstrukturen in JSON angelegt werden, ohne das zu diesem Zweck neue Tabellen per SQL erstellt werden müssen. Statt der JSON-Kodierung könnten die Daten auch relational in speziellen Tabellen gespeichert werden, die dann per JOIN-Operationen oder weitere (ggf. nested) SELECT-Operationen ausgelesen, gefiltert und sortiert werden.
In der aktuellen MySQL-Version 5.5 [Ora12] gibt es mittlerweile XML-Funktionen, um auf Tabellenfelder, in denen XML abgelegt wurde, via XPath Abfragen auszuführen. Meines Wissens nach fehlen ähnliche Funktionen für JSON in der aktuellen MySQL-Version; zumindest ist diesbezüglich nichts in der Dokumentation zu finden. Das bedeutet, dass es schwierig sein könnte, ein DMKD-System zu finden, das die Datenbank von Tricia komplett auswerten könnte. Durch die Übersetzung in einen Standard wie XML (eine Übersetzung, die von meinem Plugin in gewissem Umfang durchgeführt werden kann) können sämtliche XML-fähige Data Mining Werkzeuge angewandt werden.
Die für dieses Plugin wichtigen Datenbank-tables sind:

- Die Tabelle wikipage_ speichert Wikiseiten, auf die über die Klasse WikiPage zugegriffen werden.
- Die Tabelle hybridpropertydefinition_ dient der Speicherung von Attributdefinitionen, die auf Objekte der Klasse HybridPropertyDefinition gemappt werden.
- Zur Speicherung von TypeTagDefinitionen wird die Tabelle typetagdefinition_ verwendet, wobei rows in dieser Tabelle auf Objekte der Klasse TypeTagDefinition gespiegelt werden.

Der Unterstrich am Ende des Tabellennamen ist dabei eine Tricia-interne Konvention[2]. Um bei einem WikiPage-Objekt auf HybridWiki-Funktionalität zuzugreifen, steht die Funktion `adapt(Hybrid.class)` zur Verfügung.

Views und das Tricia-Template-System

View-seitig wird ein spezielles Tricia-Template-System angewendet. Dabei werden in der Tricia-Template-Datei an der gewünschten Stelle Platzhalter angebracht. Diese werden durch Dollar-Zeichen markiert z. B. $placeholder$
Kommt von einem Client nun eine Anfrage, wird zur Laufzeit aus dem Template die HTML-Datei erzeugt. Dabei werden placeholders durch entsprechende Inhalte ersetzt. Die Inhalte werden an anderer Stelle durch Java-Methoden generiert. Dabei gibt es folgende Arten von substitutions, also Template-Ersetzungsmechanismen:

[2] vergleiche http://www.infoasset.de/wikis/javadoc-import-wiki/platform-documentation-objectrelationalm, aufgerufen am 24.04.2012

- print substitution
- conditional substitution
- list substitution
- template substitution

Business logic

Handler Handler fungieren als Controller und reagieren auf HTTP Requests [Mat12a]:

- Nach dem Identifizieren der Session und dem Initialisieren des client context durch die Klasse HTTP Server wird durch die Klasse HandlerDispatcher der zuständige Handler bestimmt und geladen. Dabei wird zunächst versucht, aus dem Pfad den Handler zu bestimmen; funktioniert das nicht, so werden die in HandlerPattern definierten Regex-Pfade überprüft. Diese HandlerPattern ermöglichen lesbare URLs (im Gegensatz zu für den Laien„kryptischen“ Parametern, die sonst nach dem Fragezeichen kommen).
- Die überschreibbare Handler-Methode `checkAccess()` ist verantwortlich dafür, die Zugriffsberechtigung zu prüfen und ggf. den Zugriff zu verweigern. Auf den Benutzer kann anhand der zuvor identifizierten Session geschlossen werden.
- Die Handler-Methode `doBusinessLogic()` enthält business logic und gibt im Anschluss eine Station zurück, die die HTTP-Response (an den anfragenden Web-Browser) zurückschickt, also die Browser-Anfrage beantwortet. Alternativ kann die Station die Anfrage auch an einen anderen Handler weiterleiten.

Zugriffsrechte Um Entities vor unberechtigtem Zugriff zu schützen, haben Entities „editors“ und „viewers“, die jeweils Benutzer/Gruppen sein können. Neben frei erstellbaren Benutzern und Gruppen gibt es die built-in Gruppen, sogenannte Pseudogruppen: „Registered Users“ und „Everybody“ erlauben den Zugriff für alle registrierten Benutzer bzw. für alle; Administratoren können darüber hinaus Servereinstellungen verändern [3]. Zugriffsrechte von Entities werden standardmäßig von der jeweiligen parent Entity übernommen, können aber dann überschrieben werden.

Um die Tricia-Funktionalität zu erweitern, kann ein Plugin geschrieben werden. Ein Plugin in Tricia ist eine modulare Erweiterung der Software, die mit Hilfe von sogenannten extension points Zusatzfunktionalität zur Verfügung stellt, ohne dass Änderungen an bereits existierenden Plugins benötigt werden. Plugins können Abhängigkeiten zu anderen Plugins haben.

[3] siehe http://www.infoasset.de/wikis/hilfe/vordefinierte-gruppen (Aufgerufen am 24.04.2012)

3.2 Changeset

Bei Änderungen an PersistentEntities, die das Interface Versionable implementieren, wird über sämtliche Änderungen „Buch geführt“, damit später ein früherer Zustand wiederhergestellt werden kann, indem Änderungen rückgängig gemacht werden. Bei jeder Änderung werden Änderungen gruppiert und in der Klasse ChangeSet als Set von Objekten der Klasse Change gespeichert. Jedes Change-Objekt implementiert die Methode `undo()`, mit deren Hilfe die Änderungen rückgängig gemacht werden können. Um allerdings bei der Analyse keine tatsächlichen Änderungen vorzunehmen, wird vor dem Aufruf der `undo()`-Methode eine Kopie durch die Funktion `createWritableCopy()` erstellt.

3.3 Strukturierungstechniken

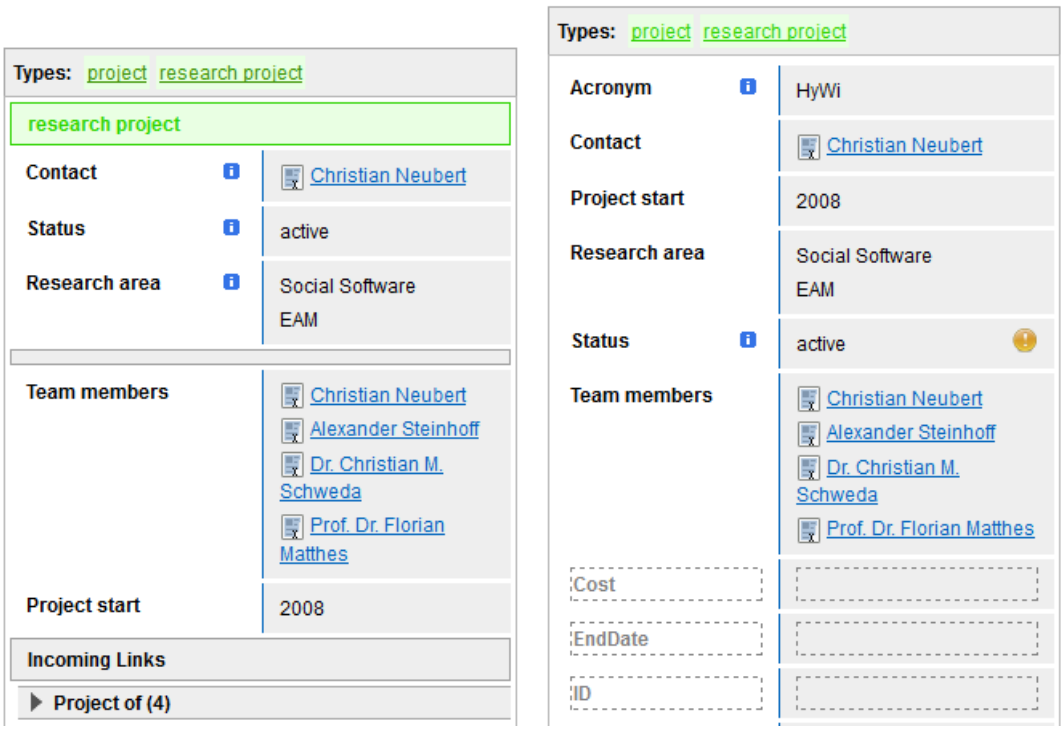
Strukturierungskonzepte HybridWikis ermöglichen eine Strukturierung von Inhalten im bottom-up-Verfahren, wobei Information im Gegensatz zu semantischen Wikis auch ohne Ontologie strukturiert gespeichert werden kann. Eine Art „Ontologie“ kann auch zu einem späteren Zeitpunkt (optional) durch Einführung von Integritätsbedingungen erstellt werden, so dass eine inkrementelle Evolution der Struktur bei HybridWikis vorgesehen und möglich ist. Damit liegt der Fokus von HybridWikis auf Flexibilität im Umgang mit strukturierten Informationen.[Neu12].

Ich möchte nun kurz Abb. 3.1 beschreiben: Wie man in (a) sehen kann, stehen in der obersten Zeile die Typen project und research project. Die Attribute Contact, Status und Research Area gehören dabei zum Typ research project und besitzen Integritätsbedingungen (sichtbar an der blauen Box mit weißem i), während die übrigen Attribute keine Definition besitzen. Dieser Snapshot stammt von der Hybrid-Wiki-Seite des sebis Lehrstuhls.

(b) zeigt das Aussehen der Box auf der gleichen Seite zu Beginn meiner Bachelorarbeit. Man sieht, dass die Gruppierung nach Typ noch nicht implementiert war. Das ist ein wichtiger Hinweis, da die neue Gruppierungsfunktion einen weiteren Anreiz schafft, Typdefinitionen anzulegen, so dass diese Änderung zu einer Veränderung im Nutzungsverhalten beim Editieren führen könnte. Das könnte sich darin zeigen, dass etwa mehr Attributdefinitionen angelegt werden, um die Gruppierungsfunktion verwenden zu können. Um Daten strukturiert abzuspeichern, verfügen HybridWikis über folgende Strukturierungskonzepte:

- Typen (sogenannte type tags) und Typdefinitionen
- Attribute und Attributdefinitionen

Typdefinitionen „definieren“ die Verwendung von type tags genauer, um deren konsistene Verwendung zu erzwingen, die den Nutzen von type tags erhöhen. Typen mit Typdefinition werden auch strukturierte Typen genannt[Mat11a].
Attribute haben jeweils einen Attributtyp. Um für eine einheitliche Nutzung von Attributen zu sorgen, können Attributdefinitionen entweder den Attributtyp festlegen oder die



(a) Attribute und Typen der Wikiseite Hybrid Wikis der sebis-Lehrstuhl-Website vom 03.05.2012

(b) Zum Vergleich die selbe Box zum Beginn meiner Bachelorarbeit.

Abbildung 3.1: Attribute und Typen zu Wikiseiten werden in einer Box rechts vom Text dargestellt. In (b) sieht man außerdem die Attributnamen-Vorschläge.

Multiplizität beschränken. Dabei kann auf folgende in Tricia implementierten Attributtypen zurückgegriffen werden[Mat11a]:

- string: Standardmäßig werden Attributwerte als Text/String aufgefasst.
- link: Links zu anderen Wiki-Seiten oder auch zu externen Seiten, ggf. mit target constraint. Unter target constraints versteht man, dass die verlinkte Seite gewisse type tags besitzen muss.
- date: Datum, das auch mit Hilfe eines Kalenders eingetragen werden kann.
- enumeration: Auflistung von mehreren String-Werten
- number: Zahlenwert (integer oder double), wobei die Nachkommastellen intern als precision gespeichert werden.
- percentage: Zahlenwert von 0-100, der mit einem % ausgegeben wird. Dieser Datentyp fand allerdings kaum Verwendung und wurde deshalb gegen Ende der Bearbeitungszeit meiner Bachelorarbeit von den Entwicklern des HybridWikis gelöscht.
- currency: Preisbeträge in verschiedenen Währungen, wobei die Standardwährung Euro ist.

- boolean: Wahrheitswerte true oder false, die als Haken bzw. kein Haken in einer Box dargestellt werden.

Die in der Attributdefinition angegebenen Beschränkungen bezüglich des Attributtyps erlauben auch, subtypes des geforderten Attributtyps zu verwenden. Beispielsweise kann statt eines Strings auch ein Datum gespeichert werden, da das Datum ein subtype von String ist. Gleiches gilt auch für Boolean-Werte, die subtype von String sind.
Des Weiteren kann mit Hilfe von Multiplizitätsbeschränkungen die „Mehrfachheit" von Attributen festgelegt werden. Dabei stehen keine Einschränkung (0..*), Exactly one value (1), At most one value (0..1) und At least one value (*) zur Auswahl.

Attributdefinitionen können als strikt oder nicht strikt festgelegt werden. Strikte Attributdefinitionen verbieten es, Attribute anzulegen, die nicht den von der Definition geforderten Kriterien entsprechen. Nicht-strikte Attributdefinitionen ermöglichen es, auch Attribute anzulegen, die laut Attributdefinition nicht erlaubt sind. Zwar wird in der Attributliste graphisch angezeigt, dass es sich um ein ungültiges Attribut handelt, allerdings wird der Nutzer nicht davon abgehalten ein solches Attribut zu speichern. Eine Übersicht ermöglicht es, Attribute die nicht den Regeln entsprechen zu identifizieren und auszubessern. Um eine einheitliche Verwendung zu ermöglichen und andere Nutzer über Sinn, Zweck und Anwendungsbereich des Attributnamens zu informieren, gibt es ein Beschreibungsfeld.

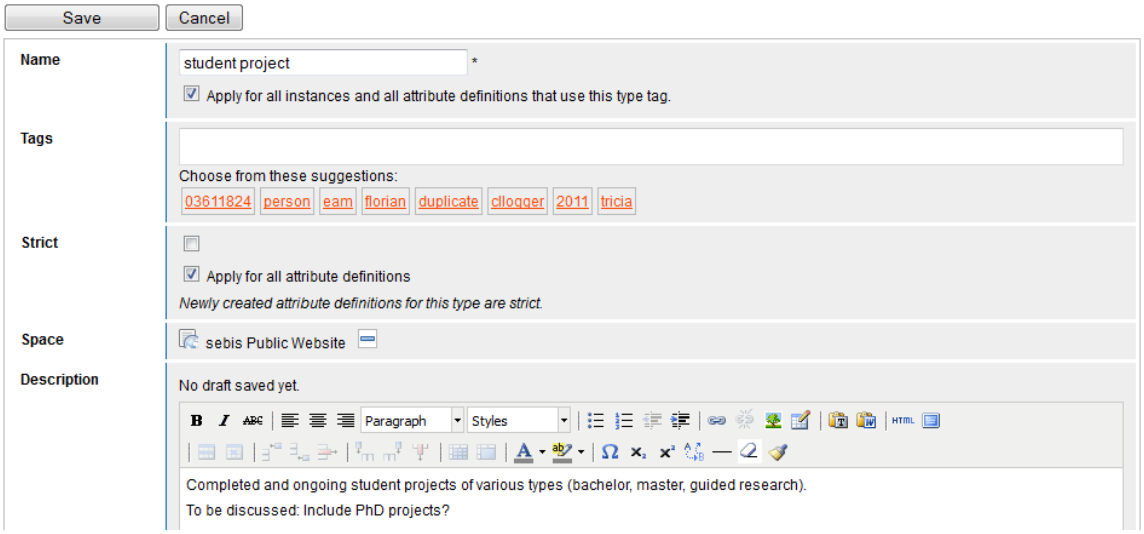


Abbildung 3.2: Screenshot beim Bearbeiten der Typdefinition student project.

Strukturierungsmechanismen Für die Arbeit mit den eben genannten Strukturierungskonzepten gibt es folgende Strukturierungsmechanismen:

- Die Autovervollständigungsfunktion schlägt dem Benutzer beim Editieren von type tags, Attributnamen und Attributwerten Werte vor, die bereits zuvor eingetragen wurden. Beim Editieren von Attributen und type tags werden – basierend auf bereits

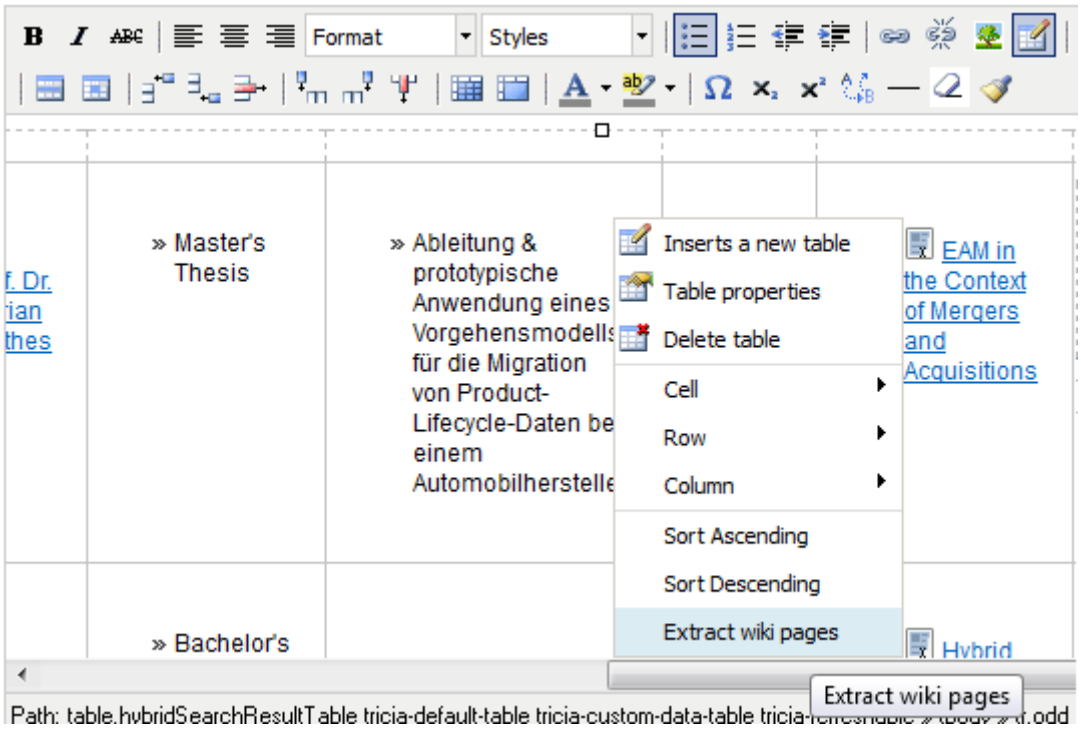


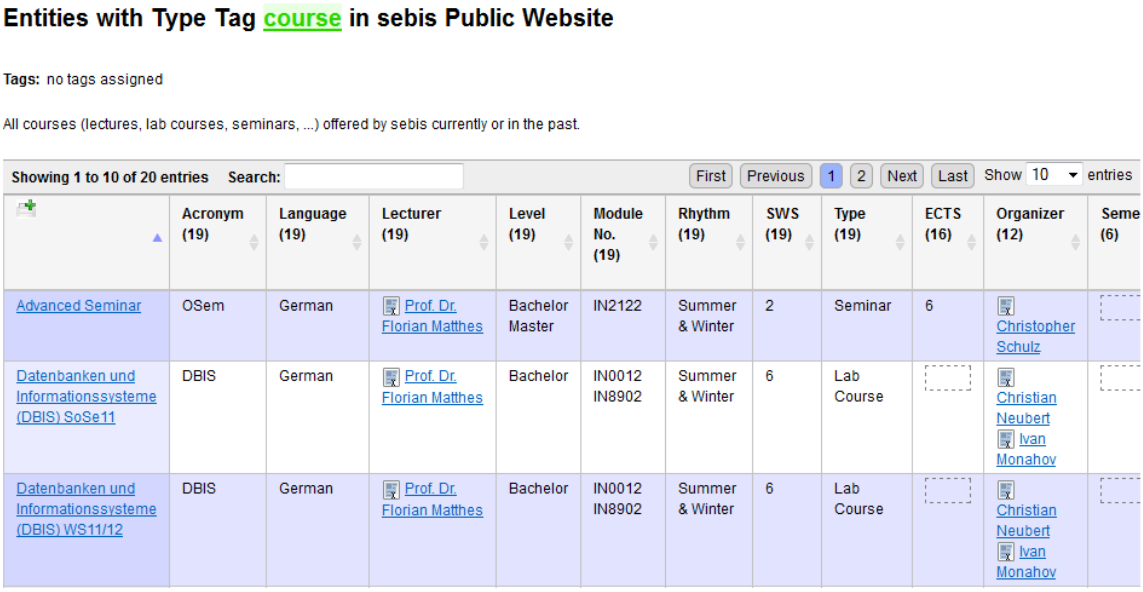
Abbildung 3.3: Kontextmenü im Richtexteditor, das den HTML2Hybrid-Mechanismus auf einer Beispieltabelle zeigt.

hinzugefügten Attributen bzw. type tags – weitere Attribute und type tags vorgeschlagen, indem „ausgegraute“ Attributfelder dafür angelegt werden. Bei der Eingabe von Attributwerten (wie Links), werden Werte (aber auch Aktionen, wie Extract wiki page vorgeschlagen. Im Fall von Link-Vorschlägen hilft die Vorschlagfunktion nicht nur dabei, die richtige Seite zu finden; bei vorgeschlagenen Links kann man auch davon ausgehen, dass die verlinkten Wikiseiten wirklich existieren.

- Extract wiki page (Attributwerte): Wiki-Seiten werden aus Attributwerten erstellt und verwandeln den Attributwert in einem Link auf die neu erstellte Seite.
- Der „Apply for all instances“-Mechanismus dient der Konsolidierung und findet Verwendung, wenn eine Definition für type tags oder Attribute (also eine Integritätsbedingung) für alle Instanzen gelten soll.
- HTML2Hybrid: Wird im Richtexteditor einer Wikiseite das Kontextmenü zu einer Tabelle geöffnet, so steht auch die Option „Extract wiki pages“ zur Verfügung. Dabei werden aus einer Tabelle die Attribute extrahiert und anschließend wird vorgeschlagen, diese Attribute anzulegen.

Außerdem stehen in Tricia verschiedene Möglichkeiten zur Verfügung, nach type tags zu navigieren. Mit der Suche können bereits sämtliche Wikiseiten angezeigt werden, die ein bestimmtes oder mehrere type tags enthalten. Zusätzlich stellt der typeTagDefinition-Handler (siehe Abb. 3.3) die Funktionalität zur Verfügung, um eine Tabelle mit allen Wi-

kiPages und deren Attribute zu erstellen. Diese Tabelle kann durchsucht werden und in-place editiert werden.



Entities with Type Tag course in sebis Public Website

Tags: no tags assigned

All courses (lectures, lab courses, seminars, ...) offered by sebis currently or in the past.

Showing 1 to 10 of 20 entries Search: First Previous 1 2 Next Last Show 10 entries

Acronym (19)	Language (19)	Lecturer (19)	Level (19)	Module No. (19)	Rhythm (19)	SWS (19)	Type (19)	ECTS (16)	Organizer (12)	Seme (6)
Advanced Seminar	OSem	German	Prof. Dr. Florian Matthes	Bachelor Master	IN2122	Summer & Winter	2	Seminar	6	Christopher Schulz	
Datenbanken und Informationssysteme (DBIS) SoSe11	DBIS	German	Prof. Dr. Florian Matthes	Bachelor	IN0012 IN8902	Summer & Winter	6	Lab Course		Christian Neubert Ivan Monahov	
Datenbanken und Informationssysteme (DBIS) WS11/12	DBIS	German	Prof. Dr. Florian Matthes	Bachelor	IN0012 IN8902	Summer & Winter	6	Lab Course		Christian Neubert Ivan Monahov	

Abbildung 3.4: Hier wird Typübersicht zu dem Typ course angezeigt. Diese Tabelle kann auch bearbeitet werden, wobei beim Editieren ein ChangeSet angelegt wird, das auch Angaben über den verantwortlichen Handler (also typeTagDefinition) enthält. Daraus kann bestimmt werden, wie häufig dieser Handler zum Editieren verwendet wurde (was auch mit meinem Plugin möglich sein wird).

4 Analyse

In der Regel wird bei der Projektanalyse eine sehr abstrakte, technologieunabhängige Perspektive eingenommen; da bei dieser Bachelorarbeit die Strukturierungstechniken eines bestehenden Systems zu untersuchen sind, ist allerdings schon im Vorfeld die zu verwendene Technologie festgelegt. Die Funktionalität der Software, die im Rahmen dieser Bachelorarbeit entstehen soll, orientiert sich an den im Folgenden beschriebenen Forschungsfragen.

4.1 Forschungsfragen

Diese Forschungsfragen sind in Zusammenarbeit mit meinem Betreuer, Christian Neubert, entstanden:

- q1: Wie werden die Strukturierungskonzepte und -mechanismen von Hybrid Wikis verwendet?
- q2: Wie können die Messergebnisse verwendet werden, so dass die Nutzer von Hybrid Wikis davon profitieren?

Hypothesen Im Folgenden sollen zahlreiche Hypothesen aufgestellt werden.

- H1: Hybrid Wiki Strukturierungskonzepte werden verwendet.
- H2: Hybrid Wiki Strukturierungsmechanismen werden verwendet.
- H3: Struktur entsteht inkrementell, d.h. der Strukturkern um Attribute und Relationen entsteht schrittweise über einen längeren Zeitraum hinweg und nicht innerhalb kurzer Zeit.
- H4.1: Hybrid Wiki Mechanismen führen zu einer Strukturkonvergenz, also zu einer einheitlicheren Verwendung der Struktur.
- H4.2: Konzepte „erhärten“ mit der Zeit, d.h. werden unmittelbar nach dem Entstehen stark verändert, nach längerem Bestehen hingegen nur noch wenig.
- H5: die Strukturierungsmechanismen von Hybrid Wikis werden genutzt, um vom Unstrukturierten zum Strukturierten zu gelangen bzw. die Qualität/Quantiät der Struktur zu erhöhen.
- H6: Attribute und Typen werden unabhängig voneinander genutzt.
  - Es gibt Seiten, die nur Attribute besitzen.

  - Es gibt Seiten, die nur type tags besitzen.
- H6: Attribute und Typen werden unabhängig voneinander genutzt.
  - Es gibt Seiten, die nur Attribute besitzen.
  - Es gibt Seiten, die nur type tags besitzen.
- H7: Struktur bringt unmittelbaren Nutzen für Benutzer.
- H7.1: Verwendung der type tag Übersicht zur Navigation und zum Editieren (default views)
- H7.2: Suche: Attribute und Typen werden in eingebetteten Suchen verwendet. Es wird mit Hilfe der Attributfacette gesucht. Es wird mit Hilfe der type (tag) Facette gesucht.

Quantitative Analyse Einblicke in die Verwendung von hybriden Wikis geben auch folgende Kennzahlen:

- Anzahl der Wikiseiten mit Attributen bzw. deren Anteil an allen Wikiseiten
- Anzahl der Wikiseiten mit Typen bzw. deren Anteil an allen Wikiseiten
- Anzahl der Wikiseiten mit Attributen und Typen
- Anzahl der Wikiseiten mit Attributen ohne Typen
- Anzahl der Wikiseiten ohne Attributen und Typen
- Anzahl Attribute pro Wikiseite (anfangs und am Ende)
- Anzahl Typen pro Wikiseite (anfangs und am Ende)
- Anzahl Attribute pro Typ (anfangs und am Ende)
- Anzahl Wikiseiten, die zuerst einen Typ haben
- Anzahl Wikiseiten, die zuerst ein Attribut haben

Fragen, die das Schema betreffen:

- Anzahl der Typen mit Typdefinitionen
- Anzahl der strikten Typdefinitionen
- Anzahl der Typdefinitionen mit Beschreibung
- Anzahl der Attribute mit Attributdefinitionen
- Anzahl der strikten Attributdefinitionen
- Anzahl der Attributdefinitionen mit Beschreibung
- Anzahl der Attributdefinitionen mit Multiplizitätsbeschränkungen (*, 1, 0..1)

- Anzahl der Attributdefinitionen mit Typen vom Typ String, Link, etc.

Dabei sollten die Forschungsfragen als „Wegweiser“ für das Plugin dienen; das Plugin sollte darüber hinaus auch noch Nachfragen explorativer Natur sowie die Bestätigung weiterer Hypothesen ermöglichen. Immerhin gibt es Fragen, die sich erst bei der Auswertung der Daten ergeben, etwa: Um welche Wikiseiten handelt es sich, die ein bestimmtes „Verhalten aufweisen“ und worin bestehen ihre Ähnlichkeiten?

4.2 Use cases

Die stakeholder, für die meine Implementierung gedacht ist, sind:

- die Entwickler der Hybrid Wikis, die mit diesem Tool Einblick in die Nutzung der Strukturierungstechniken erhalten;
- alle, die sich wissenschaftlich mit dem Thema Hybrid Wikis und deren Evolution auseinandersetzen;
- Administratoren, die sich einen Gesamteindruck über das eigene Wiki verschaffen möchten.

Der Grund dafür, dass der Standardnutzer von Tricia nicht in der Liste der stakeholder aufgeführt ist, liegt darin, dass die Nutzung von hybriden Wikis die Kenntnis von Nutzungsdaten nicht voraussetzt und die Nutzungsdaten in gewisser Weise von privater oder überwachender Natur sind. Immerhin wird beschrieben, zu welcher Uhrzeit der Editor einer Seite welches Strukturierungselement angelegt hat.
Für jede der in der obigen Liste aufgeführten Benutzergruppen ergeben sich bestimmte Anforderungen. Allen Benutzergruppen ist gemeinsam, dass die jeweiligen Fragen zur Evolution entweder durch Standardvisualisierungen bereits abgedeckt sein können oder weitere Nachfragen nötig sind. Das bedeutet, dass man einen Teil der Fragen vorhersehen und deshalb „vorsorglich“ beantworten kann, während andere Fragen sich erst später ergeben (z. B. beim Sichten der Standardvisualisierungen), so dass eine Möglichkeit gefunden werden sollte, Nachfragen zu ermöglichen, ohne dafür Teile der Software neu schreiben zu müssen. Insbesondere sollten die Nachfragen dynamisch ablaufen können ohne die Software neu starten zu müssen und das beobachtete System dabei nicht verändern, da es andernfalls die eigenen Messergebnisse verfälschen würde.
Außerdem ist es vorteilhaft, wenn für die Nutzung der Analyse kein oder nur ein geringer Installationsaufwand nötig ist, damit das Tool plug-in fähig ist.

4.3 Möglichkeiten der automatisierten Messung der Verwendung von Strukturierungstechniken

Als nächstes stellt sich die Frage, wie kann die Verwendung von Strukturierungstechniken in Hybrid Wikis automatisiert gemessen werden kann. Dabei standen folgende Quellen zur Verfügung:

- Logdateien

- Piwik
- SQL
- Java Klassen/Objekte von Tricia

Als Anforderungen an eine geeignet Datenquelle werde ich nun eine Auswahl repräsentativer Fragen formulieren, die eine geeignete Datenquelle zu beantworten in der Lage sein muss. Diese Fragen schätze ich als elementar ein, um Fragen zur Evolution der primären Strukturierungskonzepte (Attribute, type tags sowie jeweils deren Definitionen) beantworten zu können:

- Welche Attributnamen, -werte existieren zu einem bestimmten Zeitpunkt in einer Wikiseite und welche Attributtypen (String, Link etc.) wurden verwendet?
- Welche type tags existierten zu einem bestimmten Zeitpunkt in einer Wikiseite?
- Existieren zu den Attributen bzw. type tags Definitionen und wie sehen sie aus?
- Welche Änderungen wurden an einer Wikiseite zwischen Zeitpunkt t_1 und Zeitpunkt t_2 vorgenommen?

Um Abschätzen zu können, welche der jeweiligen Datenquellen geeignet sind um eine automatisierte Analyse zu ermöglichen sollen im Folgenden die einzelnen Datenquellen genauer erläutert werden.

Logdateien Bei Logdateien handelt es sich um Textdateien, die alle Anfragen an den Server protokollieren.
Üblich ist eine Zeile pro Anfrage, die

- den Zeitpunkt der Anfrage,
- die Request-Art (POST oder GET),
- die Serverantwort (z.B. 200 für OK oder 404 Not Found),
- den Browser-String, der Angaben zum Browser enthält,
- sowie der HTTP-Referer, also die Ursprungsseite über die der Besucher auf die aktuelle Seite gekommen ist

Beispiel aus der Logdatei meiner Tricia-Installation:

```
0:0:0:0:0:0:0:1 - - [26/Apr/2012:20:11:03 +0000] "GET
/wikis/sebis/home HTTP/1.1"200 0
```

Bewertung Die Logdateien ergeben den umfassendsten Einblick in die verwendeten Handler, da die Handler als Teil der URL auftauchen. Dabei ist es sowohl möglich, die Evolution der Seiten (Editiervorgänge) als auch die Nutzung der Seiten zu erfassen, um beispielsweise Aussagen darüber zu treffen, wie veränderte Strukturierungstechniken sich auf die Besucherströme auswirken. Eine Analyse der Logdatei wäre beispielsweise mittels regulärer Ausdrücke möglich, ist allerdings aus datenschutzrechtlichen Gesichtspunkten problematisch und darüber hinaus ist es nicht möglich herauszufinden, welche Strukturierungskonzepte im Einzelnen angelegt, bearbeitet oder entfernt wurden: Man kann herausfinden welche Handler benutzt wurden sowie sämtliche Informationen die per HTTP-GET als Parameter übergeben wurden (z. B. UID), allerdings fehlen die HTTP-POST Inhalte, also zum Beispiel Formularinhalte. Da es aber die Formularinhalte sind, die die Strukturierungstechniken widerspiegeln, ist die Verwendung von Logdateien für die Visualisierung der Evolution von Wikiseiten sowie für die Beantwortung der zuvor aufgestellten Forschungsfragen nicht zielführend.

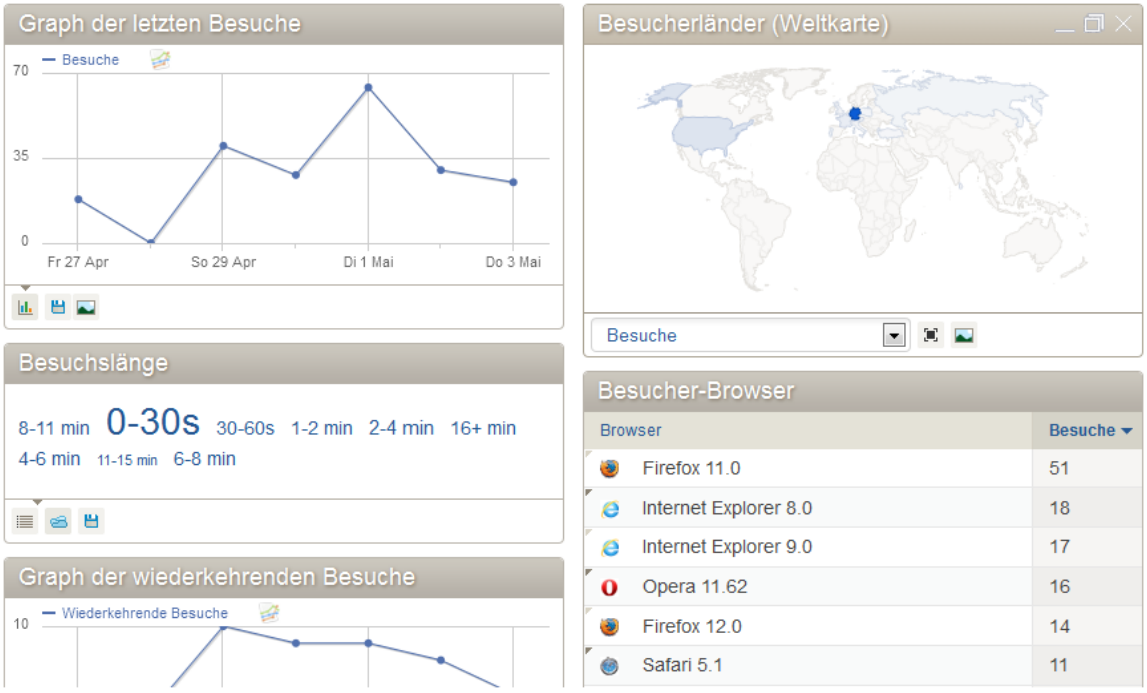


Abbildung 4.1: Dieser Snapshot meiner eigenen Piwik-Test-Installation zeigt die Visualisierung von Besucherzahlen, Besucherländer und weitere.

Piwik Piwik[1] ist eine open source web analytics software, die zahlreiche Informationen zur Nutzung der Website sammelt und diese den Website-Betreibern visuell darstellt. Dazu zählen Angaben wie durch welchen Suchbegriff und mit welcher Suchmaschine der Besucher auf die Website gekommen ist. Auch Angaben zum Ursprung des Besuchers werden auf einer Karte dargestellt. Des Weiteren können die am häufigsten besuchten Seiten

[1]Quelle: piwik.de, eigene Piwik-Installation

der Website dargestellt werden. Zum Sammeln der Daten wird auf der Website auf jeder zu „überwachenden Seite“ ein sogenannter JavaScript Tracking-Code eingebettet. Dieser Code ruft dann ein PHP-Script auf, das Nutzungsdaten in einer MySQL-Datenbank ablegt. Aus datenschutzrechtlichen Gründen kann eine Funktion zur Anonymisierung der IP verwendet werden, damit auf dem Server die IP-Adresse nicht im Klartext gespeichert wird um den den deutschen Datenschutzbestimmungen zu genügen.

Bewertung Die Daten, die Piwik speichert ähneln im Großen und Ganzen den Daten, die auch in der Logdatei stehen; allerdings enthält Piwik außerdem eine Visualierungskomponente. So eignet sich Piwik auch, um verwendete Handler zu identifizieren (ebenfalls per REGEX pattern matching), was allerdings wiederum keine Aussage darüber trifft, welche Strukturierungskonzepte im Speziellen modifiziert wurden. Aus diesem Grund die Verwendung von Piwik für die Analyse der Evolution der Strukturierungstechniken nicht zielführend, kann aber wertvolle Hinweise auf die Nutzung von Strukturierungstechniken (z. B. in Suche, type tag Übersichten) auf der Leserseite geben. Inspirierend ist Piwik allerdings für die Visualisierung der gewonnen Daten in einem Dashboard, was auch für diese Bachelorarbeit sinnvoll ist.

SQL Mittels SQL können direkt auf der Datenbank die jeweiligen Wikiseiten, Attributdefinitionen und Typdefinitionen ausgelesen werden und mit Hilfe der ChangeSets die Evolution rekonstruiert werden.

Bewertung Abfragen per SQL sind möglicherweise die performanteste Methode, um automatisierte Messungen im großen Stil in Bezug auf die Nutzung von Strukturierungstechniken zu ermöglichen. Immerhin können per SQL direkt ohne den overhead von Objektinstanziierungen Ergebnisse gewonnen werden. Das ist aufgrund der inherenten Zeitintensität der Abfragen auch sehr vorteilhaft, wenn man bedenkt, dass für jede Wikiseite sämtliche Changes einzeln rückgängig gemacht werden müssen – und das gegebenenfalls auf einem gesamten Wikispace. Vorteil dieser entwicklungstechnischen Vorgehensweise ist, dass man ohne viel Einarbeitungszeit in die Klassenstruktur relativ schnell Ergebnisse erhält. Nachteilig ist allerdings, dass in Tricia die Daten nicht rein-relational abgespeichert werden, sondern auch JSON-kodierte columns verwendet werden, so dass man entweder eigene Funktionen entwickelt (die das gleiche erledigen wie bereits in Tricia implementierte Funktionen) oder besser direkt über die jeweiligen Klassen geht.

Java-Klassen und -Objekte Die selben Daten, über die mittels SQL zugegriffen werden kann, stehen auch als Klassen zur Verfügung.

Bewertung Die Verwendung der Java-Klassen ist sicherlich die eleganteste Methode, um unabhängig von einer Datenbank eine automatisierte Nutzungsanalyse zu ermöglichen. Nachteilig war zum Zeitpunkt der Entwicklung, dass die entsprechenden Klassen, die für die automatisierte Strukturierungstechnik-Analyse nötig waren, noch keine Dokumentation im Quelltext existierte. Während man in SQL direkt sieht, welche Daten in welchem column gespeichert sind und sich somit die Bedeutung intuitiv erschließt, sieht man bei

Klassen nicht auf den ersten Blick, was mit dem jeweiligen Feld gemeint ist (z. B. context für verantwortlichen Handler).

Fazit Als Fazit kann man sagen, dass für die Analyse der Evolution der Strukturierungstechniken nur die SQL-basierte bzw. noch besser die klassen- und objektbasierte Herangehensweise sämtliche Anforderungen erfüllen, die eingangs formuliert wurden.

4.4 Beantwortung von benutzerdefinierten Abfragen

Da nun geeignet Datenquellen ausgemacht wurden, stellt sich nun die Frage, auf welche Weise diese Datenquellen genutzt werden sollten, um die zu Beginn dieses Kapitels definierten Anforderungen zu erfüllen. Die Anforderungen waren es, Standardvisualisierungen zur Verfügung zu stellen und darüber hinaus auch noch Anfragen des Benutzers zu beantworten, die durch die Standardvisualisierungen nicht abgedeckt wurden.
Eine Möglichkeit wäre es, mittels eines Konfigurators individuelle Anfragen des Benutzers abzudecken. Allerdings ist das Spektrum der denkbaren Anfragen zu groß, als dass damit jeder Fall sinnvoll abgedeckt werden könnte.
Eine andere Möglichkeit ist es, eine Abfragesprache zu verwenden, die Anfragen dieser Art beantworten hilft.
Jede Abfragesprache operiert auf einer bestimmten Datenstruktur, die mittels Queries ausgelesen werden kann. Ich werde mich dabei auf Datenstrukturen in Baum-Form beziehen, weil Bäume hierarchische Abhängigkeiten am besten abbilden können und somit für den gegeben Fall eine ideale Datenstruktur sind. Gleichzeitig kann eine Datenstruktur triviale Abfragen nur aus einer „Perspektive“ zulassen, während bei anderen Abfragen zusätzliche Logik implementiert werden muss.
Im Folgenden möchte ich einige dieser „Perspektive“ vorstellen und anschließend bewerten:

Die Wikiseiten-spezifische Sicht: Bei der Wikiseiten-spezifischen Sicht ist eine Wikiseite das ordnende Element. Bei dieser Sicht wird die Evolution jeweils pro Wikiseite aufgetragen. Diese Datenstruktur ist prädestiniert für Abfragen der Art:

- Wie viele Wikiseiten hatten von Anfang an in ihrer Evolution Typen vor Attributen (bzw. vice versa)?
- Wie viele Wikiseiten hatten im Laufe ihrer Evolution immer mehr Attribute als Typen?

Die Attribut bzw. Typ-spezifische Sicht sowie die Definitionsspezifische Sicht: Bei dieser Sicht sind die verwendeten Elemente das sortierende Element. Für die jeweils verwendeten Strukurierungskonzepte (Attribute, Typen, Attributdefinitionen, Typdefinitionen) wird jeweils angegeben, in welcher Wikiseite die Strukturierungskonzepte Verwendung fanden und in welchem Zeitraum sie dabei Verwendung fanden (also Zeitraum vom Erstellen bis zum Löschen bzw. Bearbeiten des Strukturierungselements).
Diese Datenstruktur eignet sich vor allem, um Abfragen dieser Art:

- „Wie oft wird das Attribut Acronym zum Zeitpunkt t_1 verwendet?“
- „Wie oft wird der Typ research project zum Zeitpunkt t_2 verwendet?“

Bei den aufgestellten Forschungsfragen ist aufgefallen, das Wikiseiten-spezifische Abfragen häufig auftreten und deshalb auf jeden Fall unterstützt werden sollten. Insbesondere hat sich herausgestellt, dass Wikiseiten-spezifische Frage sehr facettenreich sein können (z. B. Wikiseiten die zuerst Attribute von Typ Link hatten; Anzahl der WikiSeiten mit aktiviertem TriciaScript, die Attribute haben/nicht haben; Anzahl der Attribute pro Textzeichen im RichText), so dass Nachfragen von Wikiseiten-spezifischer Natur vielfältig, kaum standardisierbar und deshalb besondere Unterstützung erhalten sollten.

Beim Abwägen der beiden Methoden, also der Verwendung einer Wikiseiten-spezifischen Datenstruktur und der Strukurierungskonzept-spezifischen Datenstruktur für die Abfrage habe ich mich für erstere entschieden. Die gewählte Methode bietet meines Erachtens auch einen intuitiveren Zugang zur Evolution, da eine solche Datenstruktur innerhalb jeder Wikiseite die Evolution der Strukturierungskonzepte zeigt und so auch für Menschen lesbarer und verständlicher ist. Insbesondere, kann mit einer solchen Datenstruktur der zum jeweiligen Zustand gültige Status quo intuitiv und direkt abgelesen werden, also etwa welche Attribute und Typen zum jeweiligen Zeitpunkt vorhanden waren.

Fazit Zusammenfassend ist zu sagen, dass sich die gewählte Datenstruktur an den am häufigsten erwarteten Anfragen orientiert, allerdings ein kleines Subset von Abfragen erschwert (z. B. Anzahl der Seiten mit einem bestimmten Attribut zu Zeitpunkt t_2) Um die erschwerten Abfragen auch zu ermöglichen, werden ich diese im Dashboard selbst implementieren.

4.5 Inkrementelle Entwicklung

Bei einem Softwareprojekt stellt sich immer die Frage, nach welchem Modell die Softwareentwicklung organisiert wird. Das verwendete Modell richtet sich dabei jeweils nach der Art des Projekts.
Da bei dieser Bachelorarbeit ein Plugin für ein bestehendes System entwickelt werden soll, stand am Anfang dieser Bachelorarbeit die Einarbeitung in die Konzepte und Mechanismen, die bereits in Kapitel 2 beschrieben wurde. Nachdem nun die Nutzung von Java-Klassen als sinnvollste Variante für die Gewinnung von Nutzungsdaten identifiziert wurde, erfolgte die Einarbeitung in den bestehenden Quelltext sowie in die Datenstruktur. Der Quelltext von Tricia bzw. HybridWikis ist zum Großteil nicht kommentiert und die JavaDocs decken leider nur Teilaspekte ab, die für die Entwicklung von Plugins von Interesse sind, weshalb der Einarbeitungsprozess sehr mühsam war. Einzelne Module bieten sich an, nach anfänglicher Analyse und Design implementiert zu werden, ohne im Anschluss weitere Iterationsschritte an diesen Modulen vorzusehen. Dazu zählen beispielsweise die Klassen, die die Zustände von Wikiseiten zu verschiedenen Zeitpunkten rekonstruieren. Trotzdem kam das klassische Wasserfallmodell nicht zur Anwendung, damit das gewonnene Ergebnis im Anschluss mit Hilfe von Iterationsschritten verfeinert werden kann. Es hat sich bereits herauskristallisiert, dass ein Abfragesystem erforderlich ist,

das Abfragen auf den Nutzungsdaten zulässt. Manchmal ist es sinnvoll, die Datenstruktur nach intensiver Analyse festzulegen und dann nur noch marginal zu verfeinern oder zu erweitern. Es wird aber schwierig, wenn es darum geht wie die gewonnen Abfrageergebnisse visualisiert werden, weil es ein großes Spektrum von Visualisierungen gibt. Jede Visualisierung kann einen Teilbereich besonders gut darstellen; so kann etwa ein Kreisdiagramm Verhältnisse gut darstellen, während ein Graph gut für die die Visualisierung von Verläufen geeignet ist. Insbesondere der Teilbereich, in dem es darum geht, dem Nutzer Visualisierungen mit Hilfe der restriktiven Templatesprache Trica Script zur Verfügung zu stellen, bedarf zahlreicher Iterationen.

5 Designentscheidungen

5.1 XML und XPath als Grundlage für dynamische Abfragen

Für die automatisierte Auswertung der Nutzungsdaten sollte ein Plugin angefertigt werden. Damit das Analysetool für den Anwender von Nutzen sein kann, sollte es unter anderem auch dynamische und veränderbare Abfragen durch den Anwender ermöglichen. Die Abfragen können entweder mit Hilfe einer existierenden Abfragesprache oder durch eine eigens für diesen Zweck entwickelte Abfragesprache durchgeführt werden. Entscheidungskriterium für die Abfragesprache war dabei einfache Bedien- und Erlernbarkeit, Simplizität, um bequem und unkompliziert Anfragen an das System zu richten.
Bei der Abfragesprache SQL waren diese Kriterien nicht erfüllt. Es gibt zahlreiche Datenbanksysteme, die den SQL-Standard implementieren. Im Folgenden soll die Datenbanksoftware MySQL beispielhaft als eine SQL-Implementierung behandelt werden.

Da Tricia bei entsprechender Konfiguration auf einer MySQL-Datenbank mit der Abfragesprache SQL zur Verfügung steht, wäre es naheliegend, auf SQL zurückzugreifen. Dann könnten die Daten direkt aus der Datenbank extrahiert werden und so Abfragen erstellt werden. SQL ist als deklarative Sprache relativ einfach zu erlernen, allerdings ist es schwierig mit JOINs die entsprechenden Daten zusammenzutragen. Außerdem werden viele Daten durch das ORM-Modul in JSON abgelegt, das in aktuellen MySQL-Implementierungen (vergleiche Abschnitt 2.1) nicht per SQL abgefragt werden kann. Außerdem ist es mit SQL schwierig verschachtelte has many-Beziehungen darzustellen bzw. abzufragen, da dazu verschachtelte Abfragen nötig wären, die wiederum Logik außerhalb der SQL-Abfrage bräuchte (beispielsweise eine `while`-Schleife in einem Java-Programm). Jede SQL-Abfrage kann nur rows mit jeweils gleichbleibenden columns zurückgeben; eine Auflistung mehrerer Werte in einer columns ist nicht ohne weiteres möglich. Würde man also auf Basis einer SQL-Datenbank eine Visualisierung erstellen, müsste jede Visualisierung eigens „programmiert werden“; dynamische Abfragen wären damit kaum mehr möglich.

Die Abfragesprache XPath wird in Verbindung mit XML verwendet. XML ist nicht nur maschinenlesbar, sondern auch für Menschen lesbar.
XPath hat außerdem die Eigenschaft, dass es - syntaktische Korrektheit vorausgesetzt - keine „Fehlabfragen“ gibt. Das heißt, wenn man die Anzahl der XML-Elemente mit node name <test> sucht, es aber keinen Node bzw. Element namens „test“ gibt, kommt es zu keinem Fehler, weil das Element nicht gefunden wurde, sondern lediglich zu keinen Ergebnissen bzw. boolean false. Konkret kann das bedeuten, dass eine Teilabfrage nicht beantwortet wird, während die übrigen Teilabfragen korrekt sind und ggf. visualisiert werden. Nur durch eine Verifizierung der Abfragen von Hand kann sichergestellt werden, dass die angezeigten Ergebnisse auch den erwarteten Ergebnissen entsprechen.

5.2 Java-Packages

Das Plugin besteht aus zahlreichen Klassen, die in verschiedenen packages gruppiert wurden. Der Übersicht halber werde ich an dieser Stelle die einzelnen packages vorstellen, bevor ich in den kommenden Abschnitten Designentscheidungen erläutern werde, die sich auf diese packages beziehen.

- Das Hauptpackage des Plugins ist de.infoasset.usageAnalysis.
  Direkt in diesem Package befindet sich gemäß der Tricia-Plugin-Spezifikation die Klasse UsageAnalysisPlugin.
- Das package de.infoasset.usageAnalysis.snapshot enthält die Logik zur Generierung der XML-Datenstruktur, auf der später Abfragen möglich sein sollen. Innerhalb dieses packages gibt es noch eine Unterteilung in attributeDefinitions und typeDefinitions, die jeweils für die Rekonstruktion der Attribut- bzw. Typdefinitionen verantwortlich sind.
- Methoden für Abfragen auf der XML-Datenstruktur werden im package de.infoasset.usageAnalysis.query definiert.
- de.infoasset.usageAnalysis.cache enthält verschiedene Klassen, die mehrstufige Cachingmechanismen anbieten, um die Auslieferung der Abfrageergebnisse für TriciaScript-Funktionen zu beschleunigen.
- Helferfunktionen, die in vielen Bereichen im Code verwendet wurden, befinden sich im package de.infoasset.usageAnalysis.helpers.
- de.infoasset.usageAnalysis.handler enthält gemäß Tricia-Konvention die Handler, die von diesem Plugin zur Verfügung gestellt werden.
- Um die Wiederverwendbarkeit von implementierten Tricia-substitutions zu ermöglichen, wurden einige substitutions in das Package de.infoasset.usageAnalysis.templates ausgelagert. Die dort definierten Methoden werden introspektiv von mehreren Handlern ausgelesen und als TriciaScript-Funktion zur Verfügung gestellt.

Die Gruppierung der Klassen erfolgte auch nach dem Kriterium, jeder entwickelten Funktionalität einen bestimmten Ort zuzuweisen, um Durchsuchen von Quelltext im Entwicklungsprozess zu vermeiden, was Zeit spart. Das war durch eine klare Delegation von Zuständigkeiten an verschiedene Packages und darin wiederum an Klassen möglich.

Eine genauere Beschreibung der Klassen ist in JavaDoc zu finden, da die Klassen großzügig kommentiert wurden.

5.3 Plugin

Die Klasse UsageAnalysisPlugin ist eine Default-Klasse meines Tricia-Plugins [Mat11b] und ist für die Initialisierung der Plugin-Funktionalität zuständig. Im Fall dieses Plugins war das vor allem die Definition von Objekten der Klasse AdditionalActionExtension und AdditionalViewExtension, die introspektiv dafür sorgen, dass

- eine weitere Action „Analyze wiki page(s)“ dem SearchHandler hinzugefügt – allerdings wird die Action nur angezeigt, wenn die Suche auf Wikiseiten beschränkt ist;
- jede WikiPage eine weitere View erhält („Analyze wiki page(s)“);
- jede WikiPage eine weitere Action erhält („Clear cache“), um den Cache zu leeren und das XML-Dokument neu zu generieren;
- jedes Wiki eine weitere View erhält („Analyze Definitions“).

Templates Im Normalfall werden in einem Plugin im package handler sämtliche Handler definiert und innerhalb der Klasse auch die substitutions erstellt. Dieser Konvention bin ich nicht gefolgt; stattdessen habe ich die substitutions in das package templates ausgelagert. Dort können in den Klasse PrintSubstitution bzw. ListSubstitution Funktionen definiert werden, die – sofern sie nur einen Parameter der Klasse HandlerInterface oder eine ihrer Subklassen und den Rückgabewert String besitzen – introspektiv durch den entsprechenden Handler in `putContentBodySubstitutions()` als substitutions registriert werden. Ist dabei der Funktionsparameter

- vom Typ DefinitionsAnalysisHandler, so wird die substitution nur für den DefinitionsAnalysisHandler registriert;
- ist der Parameter vom Typ WikiPagesAnalysisHandler, so wird die substitution nur für den DefinitionsAnalysisHandler registriert;
- bei einem Funktionsparameter HandlerInterface wird die entsprechende substitution für bei beiden Handlern registriert.

Der Grund für diese Vorgehensweise ist zum einen der dadurch resultierende „sauberere“ Code mit verbesserter Lesbarkeit und weniger Zeilen-Einrückungen und zum anderen die leichtere Wiederverwendbarkeit durch andere `putSubstitutions()`-Methoden in neuen Handlern. Vor allem beim Anlegen neuer Handler-Klassen hat diese Vorgehensweise den Arbeitsaufwand für die „Übernahme“ von substitutions in einer neuen Klasse erleichtert.

5.4 Generiertes XML-Dokument

Bei der Entwicklung des Plugins wurde zunächst die XML-Struktur konzipiert. Wie im vorigen Kapitel in Abschnitt 4.4 bereits festgestellt wurde, ist für die Datenstruktur die Wikiseiten-spezifische Sicht vorteilhaft, da damit ein Großteil der Forschungsfragen mit relativ einfachen Abfragen beantwortet werden kann. Insofern war es modelltechnisch relativ leicht, den bereits im Analyseteil identifizierten Wikipage-spezifischen Ansatz designtechnisch in XML umzusetzen. Dabei wird das Konzept von snapshots verwendet, bei dem beginnend mit dem aktuellen Abbild der Wikiseite und seiner momentan verwendeten Strukturierungskonzepte snapshots angelegt werden, die den jeweiligen Zustand zum Zeitpunkt des snapshots sowie die Änderungen zwischen aufeinanderfolgenden snapshots repräsentieren. Das entstandene XML-Dokument bzw. dessen Schema werde ich nun

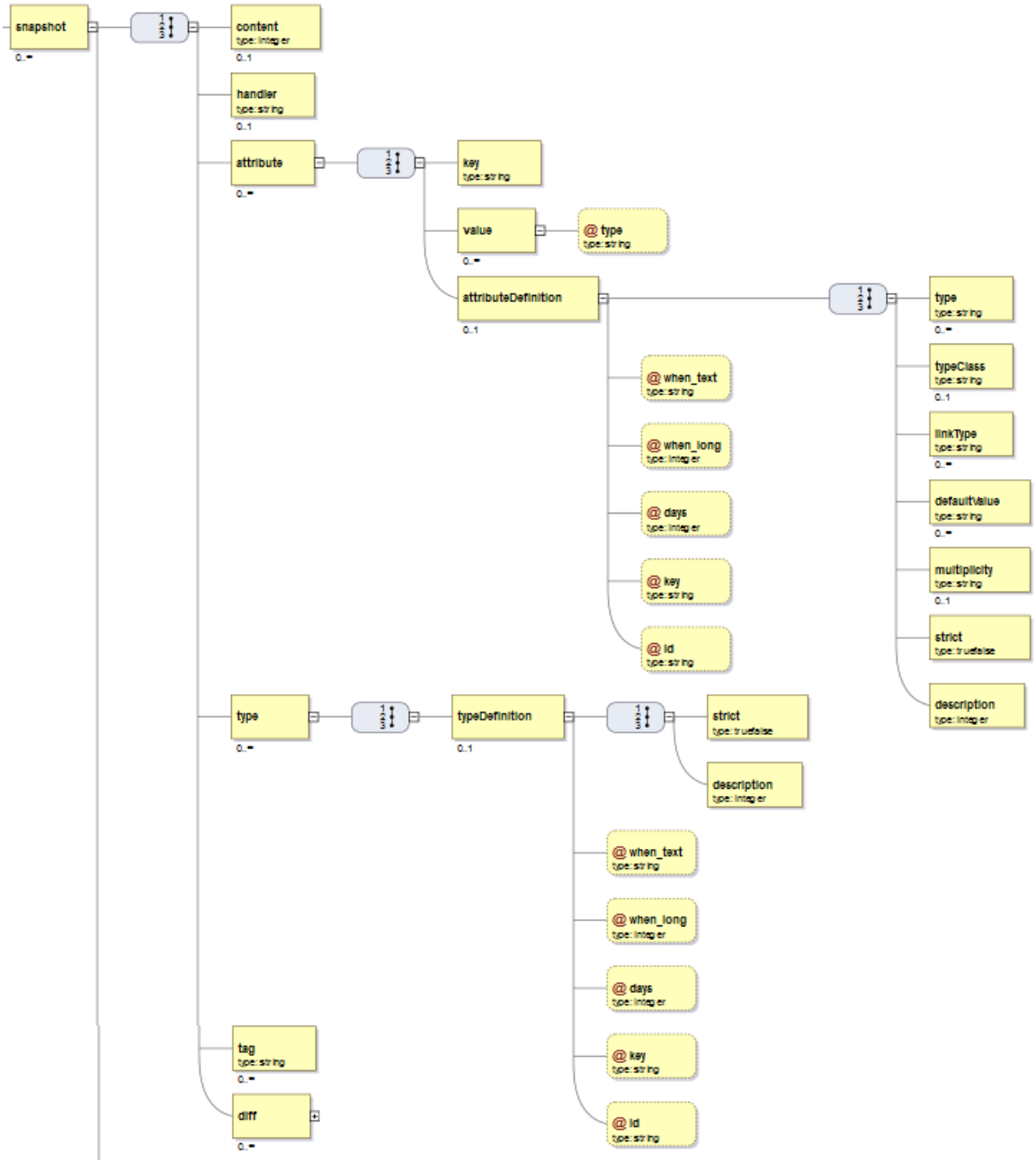


Abbildung 5.1: Visualisierung eines Auszug des XML Schemas für die generierten XML-Dokumente, graphisch nachbearbeitet. Zur Visualisierung wurde die XML-Visualisierungssoftware von Vaclav Slavětínský verwendet [Sla08].

beschreiben.
Für die folgende Beschreibung des XML-Dokuments sei darauf hingewiesen, dass mit XML-Tag die einzelnen (z. B. öffnenden und schließenden) Tags gemeint sind. Aus diesem

Grund wird der Begriff (XML-)Element konventionskonform verwendet, um sowohl auf die tags (als syntaktischer Struktur) als auch auf deren Inhalt (z. B. child nodes, text nodes etc.) zu verweisen [BPSM+08].
Die durch mein Plugin generierte XML-Datei verwendet als obligatorisches root-Element <usageAnalysis>. Innerhalb des root-Elements befinden sich sämtliche <wikipage> Elemente, die zur Analyse ausgewählt wurden. Wichtig ist, die Kleinschreibung von <wikipage> zu beachten, da XPath Klein- und Großschreibung unterscheidet.
Jedem <wikipage>-Element werden zur eindeutigen Identifizierung, zur besseren Durchsuchbarkeit und zur besseren Lesbarkeit bestimmte Attribute zugewiesen. Dazu gehören neben Attribut `id` mit der Tricia-generierten ID, dem Attribut `url` mit der relativen URL der Wikiseite auch Angaben zur letzten Änderung der Wikiseite. Die letzte Änderung stellt ein Datum dar und da des öfteren Daten dargestellt werden müssen, habe ich mich entschlossen, jedes Datum mittels drei Attributen zu kennzeichnen. Diese Attribute werden von einer speziellen Methode generiert, so dass an nur einer Stelle im Quelltext weitere Attribute definiert werden können, die das Datum darstellen (beispielsweise im Format yyyy-mm-dd, sofern eine Visualisierungssoftware das fordert).

- when_text: englische textuelle Repräsentation des Datums als String;
- when_long: long-Repräsentation des Datums (Millisekunden seit der Unix-Epoche), das verwendet wird, um ein JavaScript-Datum zu erzeugen;
- days: Anzahl der Tage zwischen heute und dem jeweiligen Datum (wobei das Datum das letzte Änderungsdatum etc. sein kann).

Diese Angaben zur letzten Änderung lassen sich zwar auch dem ersten <snapshot>-Element entnehmen, wurden aber zur Vereinfachung der Abfrageeingaben für XPath-Queries auch im <wikipage>-Element eingetragen. Jedes <wikipage>-Element enthält zu Beginn einige Zusatzinformationen, wie Text-Länge (<content>, parent WikiPage (<parent>), ggf. child WikiPages (<child>), ob Tricia-Script für die Wikiseite aktiviert ist (<scriptable>) und verwendete Tricia-Script-Funktionen innerhalb (<function>) mit den child nodes (<name>) und (<block>). Die Angaben beziehen sich dabei jeweils auf die aktuelle Wikiseite – nicht also auf Wikiseiten in der Versionshistorie – und sollen dabei helfen, Hypothesen zu testen.
Beispiel: Weisen WikiPages, die Tricia-Script-Funktionen verwenden, häufiger oder seltener Attribute oder type tags auf? Innerhalb der <wikipage>-Elemente gibt es sämtliche <snapshot>-Elemente, beginnend mit dem aktuellen Zustand, gefolgt von den snapshots der vorherigen ChangeSets. Innerhalb der <snapshot>-Elemente werden für jedes Strukturierungskonzept eigene XML-Elemente erstellt. Für jeden Attributnamen wird ein Element <attribute> mit dem child element <key> erstellt, das den Attributnamen enthält. Pro Attributwert wird ein Element <value> mit dem Attribut type angelegt, das den Attributtyp enthält. Sofern eine Attributdefinition zu dem Attributnamen angelegt wurde, wird ein XML-Element <attributeDefinition> angelegt.
Die automatisch generierte XML-Datei enthält als Elemente sämtliche durch Suche oder UID spezifizierte Wikiseiten. Jede WikiPage enthält darüber hinaus sogenannte snapshots, also Abbilder der WikiPage zum jeweiligen Zeitpunkt. Diese snapshots sind sowohl inkrementell als auch vollständig, damit der Nutzer jede erdenkliche Abfrage starten kann, egal

ob sich diese Abfrage auf den inkrementellen oder vollständigen snapshot bezieht. Das bedeutet allerdings auch, dass die XML-Datei entsprechend groß wird, da Informationen aus Gründen der besseren Lesbarkeit der XML-Datei und zur besseren Durchsuchbarkeit per selbst definierten XPath-Abfragen redundant abgelegt werden. Diese Redundanz stellt aber im praktischen Betrieb von Tricia kein Problem dar, da auf Basis dieser XML-Daten lediglich Analyse-Abfragen von (relativ) wenigen dazu autorisierten Benutzern erfolgen können.
Im Appendix ist das XML-Schema der generierten XML-Datei zu finden, das Auskunft über mögliche XML-Elemente und deren Multiplizität gibt.

5.5 Kriterien für die snapshot-Generierung

Die Frage, wann snapshots in der XML-Datei angelegt werden, ist wichtig um die gewonnen Ergebnisse richtig deuten und interpretieren zu können. Mit der bislang beschriebenen XML-Struktur wäre es möglich, jede Art von snapshot-Generierung zu verwenden. Aus performance-Gründen ist es nachteilig, aus jedem Element im ChangeSet einen Snapshot zu erstellen. Immerhin habe ich durch Sichten der ChangeSet-Tabelle festgestellt, dass es sehr viele Änderungen nur am unstrukturierten Richtext gibt, was bedeuten würde, dass sich zwischen snapshots, die nur Änderungen am strukturierten Teil der Wikiseite erfassen, keine Änderungen zeigen würden. Die daraus resultierende XML-Datei wäre für Menschen schwer zu lesen (da extrem lang mit vielen redundanten, da sich wiederholenden Informationen) und für den XPath-Evaluator sehr zeitraubend. Um die Anzahl der snapshots zu reduzieren, können verschiedene Maßnahmen verwendet werden:

- Time-Slicing-Verfahren: Änderungen, die innerhalb relativ kurzer Zeit stattfinden werden gruppiert, um die Anzahl der snapshots zu reduzieren. Bei dieser Maßnahme kann es allerdings passieren, dass kurzzeitig angelegte Strukturierungskonzepte im XML-Dokument nicht mehr auftauchen, weil sie innerhalb eines time slice angelegt und dann wieder verworfen wurden. Will man allerdings die Nutzung der Strukturierungskonzepte (Attribute, Typen) und -mechanismen (erkennbar an den verwendeten Handlern), würden beim time-slicing-Verfahren wertvolle Informationen verloren gehen, etwa welcher Handler hauptsächlich verwendet wird.

- Event-driven-Verfahren: In der objektorientierten Programmierung ist das Konzept eines EventListeners geläufig. Man kann eine Methode bei einem EventListener registrieren, die dann aufgerufen wird sobald das entsprechende event eintritt. Analog dazu kann man auch spezielle Changes, also Änderungen an den primären Strukturierungskonzepten (nämlich Attribute und Typen) als Grundlage für die snapshot-Generierung verwenden, das heißt: sobald eine Änderung an Attributen oder Typen bzw. deren Integritätsbedingungen erfolgt, wird ein neuer snapshot angelegt. Vorteil dieses Vorgehens ist, dass die Änderungen an den zu untersuchenden Objekten (Attribute und Typen) vollständig dokumentiert werden und so den Umfang des XML-Dokuments auf das Wesentliche (also Untersuchenswerte) reduziert. Änderungen an Attributen und Typen, die kurz aufeinanderfolgend getätigt wurden, werden 1:1 dokumentiert, weshalb sich die erstellte XML-Datei ideal zur Beantwortung von Fragen eignet, wie Benutzer beim Strukturieren vorgegangen sind.

Ich habe mich dafür entschieden, lediglich Änderungen an Attributen und Typen als Änderung zu definieren, während alle andern Änderungen ignoriert werden. Das bedeutet, dass nur dann, wenn es eine Änderung an Attributen oder Typen gibt, ein neues ChangeSet erstellt wird.

5.6 Template-Funktionen

Um die Ergebnisse der Analyse darzustellen (also die Ergebnisse der XPath-Abfragen auf der generierte XML-Repräsentation der Wikiseiten), stehen verschiedene TriciaScript-Funktionen zur Verfügung. Den Tricia-Script-Funktionen können entweder eine UID oder eine serialisierte JSON-Suche als Funktionsparameter übergeben werden. Wird die Funktion allerdings durch den Handler aufgerufen, können sowohl das UID als auch das JSON-Search-Argument entfallen, da dann die Angaben übernommen werden, die der Handler erhalten hat. Dieses Konzept ähnelt dem method overriding in Java insofern, als dass Standardfunktionalität (die durch die Handlerparameter vorgegeben wird) durch (konkretere) Funktionsparameter außer Kraft gesetzt werden kann. Dadurch wird die größtmögliche Flexibilität beim Erstellen des Dashboards gewährleistet:
So können etwa im Dashboard Visualisierung angezeigt werden, die sich je nach Handler-Kontext ändern, während andere Visualisierungen auf der gleichen Seite die Visualisierungen bestimmter, vorab festgelegter Wikiseiten oder bestimmter Suchanfragen darstellen. Das kann zum Beispiel dann von Vorteil sein, wenn man die analysierte Wikiseite mit einer Referenzseite vergleichen will.
Für die Visualisierung von einzelnen Wikiseiten wird ein anderes Dashboard verwendet als für die Visualisierung von mehreren Wikiseiten. Die jeweiligen Dashboards werden durch Tricia-Templates definiert.

5.7 Google Chart API

Für die Visualisierung in Dashboards wurden unter anderem auch die Google Chart API verwendet [Goo12], des weiteren wurden auch eigene Visualisierungen verwendet. Benutzerdefinierte Visualisierungen können nur auf die Google Chart API und HTML zurückgreifen.
Die Google Chart API ist eine JavaScript Bibliothek, die aus einer DataTable (Bezeichnung in der Google Chart API) eine graphische Visualisierung erstellt [1]. Die Visualisierung wird mit HTML5 und SVG-Technologie erstellt, so dass Plugins wie Flash nicht erforderlich sind. Ein großer Pluspunkt der Google Chart API ist die leichte Auswechselbarkeit der Visualisierungen sowie die große Anzahl an Visualisierungen, die unterstützt werden.
Der Download der Google Chart API, um charts auch offline zu erstellen ist nach den Google-FAQ [2] nicht zulässig. Das bedeutet, dass ohne bestehender Internetverbindung die Google-Visualisierungen nicht erstellt werden können und stattdessen leere Felder angezeigt werden. Aus diesem Grund ist bei der Verwendung des Plugins darauf zu achten, dass eine Internetverbindung besteht.

[1] https://developers.google.com/chart/interactive/docs/index (Aufgerufen am 12.04.2012)
[2] https://developers.google.com/chart/interactive/faq#localdownload (Aufgerufen am 12.04.2012)

5.8 Dashboard-Erstellung mittels JSON-Konfiguration

Ursprünglich habe ich in einem Prototyp einen JSON-String verwendet, der das Dashboard konfiguriert. Dieser bestand aus einem JSON-Array (Liste) von JSON-Objekten mit Angaben zum Titel, Höhe und Breite der chart sowie der nested XPath-Abfrage, der die DataTable für die Google Chart API erstellt. Der Vorteil dieser Konfigurationsdatei ist die knappe, präzise Schreibweise, mit deren Hilfe der nötige JavaScript-Code generiert wird, der für Google Chart API-Quelltext jedes Mal sehr ähnlich ist, da nur die DataTable sowie Legende und Bezeichnungen ausgetauscht werden. Außerdem unterstützt die Google API zahlreiche verschiedene Visualisierungen, die auf diese Weise eingebunden werden können. Allerdings hat es sich herausgestellt, das diese Konfigurationsdatei nicht die nötige Flexibilität bietet, um beliebige Visualisierungen darzustellen. Beispielsweise wäre eine Visualisierung mit HTML -Elementen nicht möglich. Solche Visualisierungen sind aber sehr praktisch, um Daten mit HTML und CSS zu visualisieren. Aus diesem Grund wurde die JSON-Konfiguration durch Tricia Template-Funktionen ersetzt.

5.9 WikiPage-Dashboard-Erstellung mittels Tricia Template

Eine Anforderung an das Plugin ist es, Funktionen zur Verfügung zu stellen, um beliebige Anfragen per XPath zu erstellen. Diese Trica-Script-Funktionen, die in jeder beliebigen WikiPage via Tricia Script eingebunden werden können, können auch innerhalb von Tricia Templates verwendet werden. Im Gegensatz zur Verwendung der Tricia-Script-Funktionen in einer WikiPage werden in der Template-Datei der Tricia-Funktion keine bestimmten WikiPage(s) übergeben. Stattdessen werden die Parameter (entweder UID oder JSON-Search-String) vom Handler übernommen. Das ist wichtig, damit das Dashboard zu den angeforderten WikiPages erstellt wird. Trotzdem ist es möglich, im Tricia Template auch eine UID zu übergeben, etwa um Referenz-Visualisierungen zu erstellen. Wird der Tricia-Funktion ein Parameter übergeben (z. B. UID), so werden die Parameter des Handlers ignoriert. Dieses Tricia-Template enthält die Standardvisualisierungen für eine einzelne WikiPage bzw. eine andere Standardvisualisierung für mehrere WikiPages. Dabei gibt es zwei Arten von XPath-Abfragen:

1. einfache XPath-Abfragen, die einen String zurückgeben (z. B. Anzahl der WikiPages mit Attributen)
2. komplexe verschachtelte XPath-Abfragen, die eine tabellenartige Struktur zurückgeben. Bei der tabellenartige Struktur handelt es sich um eine ListSubstitution, bei der für jedes per XPath abgefragte Element ein placeholder definiert wird, der dann mit dem entsprechenden XPath-Ergebnis ersetzt wird. Visualisierungsabfragen, die die soeben erwähnten Funktionen in die Google Chart API einbettet. Damit können allerdings keine Visualisierungen von Fremdanbietern dargestellt werden.

Aus einer verschachtelten XPath-Abfrage kann relativ einfach eine Tabelle erstellt werden, z. B. in Form einer HTML-Tabelle.

Datenstruktur für verschachtelte XPath-Abfragen Um beliebige verschachtelte XPath-Abfragen zu ermöglichen, habe ich eine generische Datenstruktur entwickelt mittels der

die Abfrage übergeben werden kann. Diese Datenstruktur ist ein JSONArray von JSONObjects, die jeweils zwei Schlüssel enthalten müssen:

- xpath (obligatorisch), ein String der die XPath-Abfrage darstellt; falls es sich dabei um eine Unterabfrage handelt, sollte der XPath-String relativ zur parent node geschrieben werden.
- values (obligatorisch) enthält ein JSONArray von JSONObjects mit die folgenden Schlüssel enthalten: `xpath`, `name` und `typ`.
  `xpath` gibt dabei wiederum die die XPath-Abfrage an, idealerweise relativ zum parent XPath-String.
  `name` enthält den Namen, der später als substitution zur Verfügung steht; interpretiert man das Ergebnis dieser Abfrage als Tabellenstruktur, so wäre `name` eine Spalte bzw. in MySQL-Terminologie eine column.
- nested (optional), das ein JSONArray mit der eben definierten Datenstruktur enthalten kann (oder ein leeres JSONArray).

Mit der vorliegenden XPath-Struktur kann eine Tabellenstruktur mittels beliebig verschachtelte XPath-Abfragen befüllt werden.
Eine genauere Erläuterung der Verwendung dieser Datenstruktur erfolgt im Kapitel Bedienung.

5.10 Generische list substitutions für Map-List-Datenstrukturen

Die list substitution für List<Map<String, Object>>-Objekte können mit Hilfe der generischen Tricia-Script-Funktion $list()$ erzeugt werden, die ich zu diesem Zweck entwickelt habe. Die Tricia-Funktion $list()$ hat den obligatorischen Parameter `name`, wobei `name` ein Methodenname mit bestimmter Signatur im package templates und dort in der Klasse Template sein muss.
Mit Hilfe der $list()$-Funktion können beliebige Datenstrukturen, die aus Listen bzw. arrays und maps bestehen, visualisiert werden. Beispiel für eine solche Datenstruktur ist JSON: in der Java-Implementierung von JSON gibt es die Klassen JSONArray und JSONObject, wobei JSONArray eine Java-List repräsentiert, während JSONObject als Java-Map interpretiert werden kann. Tatsächlich habe ich im helper-package in der Klasse JSONHelpers Methoden geschrieben, die eine Umwandlung von Map-List-Datenstrukturen in JSON-Datenstrukturen und vice versa vornehmen, was demonstriert, das eine verlustfreie Konvertierung zwischen den beiden Datenrepräsentationsformen möglich ist.
Die Verwendung der generischen list substitutions ist dabei folgendermaßen: Die Klasse GenericListSubstitution ist abstrakt und kann durch Implementierung der Methode `List<Map<String, Object>>get()` Daten übergeben. Dabei wird über die Elemente in der List iteriert und sämtliche keys aus der Map stehen als substitutions zur Verfügung. Generische Listen werden derzeit verwendet, um die Abfrageergebnisse bei komplexen XPath-Abfragen darzustellen, da es sich auch bei Tabellen um eine List<Map <String, Object>> handelt, über die iteriert werden soll und in einzelnen Iterationsschritten die columns eingefügt werden sollen.
Vorteil der Verwendung von generischen Listen ist zum einen weniger Quelltext als auch

die black-box-Sicht, die durch diese Klasse ermöglicht wird. Wird eine bekannte Datenstruktur vom Typ List<Map<String, Object>>übergeben, weiß man, dass bei jedem Iterationsschritt sämtliche keys der Map zur Verfügung stehen – ohne eine eigene Implementierung zu schreiben und testen zu müssen.

5.11 List und Map

Es stellt sich natürlich die Frage, was eine Umwandlung von JSON in Maps bzw. Lists bezweckt, wie sie im vorigen Abschnitt thematisiert wurde.
Meine Tricia-Script-Funktionen werden mit JSON-Strings aufgerufen, um eine entsprechend komplexe Datenstruktur zu übergeben. Diese JSON-Strings werden allerdings nicht als JSONObjects oder JSONArrays übergeben, sondern zunächst in Map bzw. List-Strukturen umgewandelt und als solche werden sie weitergegeben. Zum einen hat das den Hintergrund, dass es sich bei Map und List um interfaces handelt, und es in Java gute Praxis ist [Ull07], das Interface als Datentyp zu verwenden statt der Klasse, die das Interface implementiert (beispielsweise TreeMap, HashMap bzw. ArrayList). Allerdings ist die Verwendung von Interfaces statt den implementierenden Klassen eher ästhetischer Natur, da nicht davon auszugehen ist, beispielsweise eine HashMap statt der TreeMap verwendet werden soll. Der Grund, warum ich statt JSON auf Map/List-Datenstrukturen zurückgegriffen habe ist vielmehr, dass diese interfaces mehr Methoden definieren als die sehr einfach gehaltene JSONArray bzw. JSONObject-Klasse. Beispielsweise kennt die JSONArray-Klasse keine Methode, um zu prüfen, ob ein gewisses Objekt im JSONArray enthalten ist[3], während das interface List eine solche Methode vorsieht. Auch kann JSONArray keinen Iterator zurückgeben.
Nebeneffekt der Verwendung von Map und List ist, dass die Verwendung von JSON nicht zwingend erforderlich ist. Man könnte durch relativ wenige Modifikationen auch andere Datenstrukturen unterstützen, beispielsweise XML, das auch Listen und maps repräsentieren kann.
Alternativ zu List-Map-Strukturen wäre natürlich auch die Implementierung eigener Klassen möglich, die die Datenstruktur aufnimmt – mit expliziten Feldern für die jeweiligen Daten. Zu beachten ist allerdings, dass die Datenstrukturen dynamisch verändert werden, weil etwa das Ergebnis einer komplexen XPath-Abfrage unterschiedliche Felder hat, je nachdem welche Felder in der Abfrage spezifiziert wurden. Aus diesem Grund wäre es technisch kaum realisierbar, explizite Klassen statt der verwendeten List-Map-Datenstrukturen zu verwenden.

5.12 Caching-Mechanismen

Wenn man das Plugin ohne Caching-Mechanismen nutzt fällt auf, dass jeder Seitenaufbau relativ lange dauert[4]. Das liegt daran, dass jedes Mal für jede WikiPage die zugehörige XML-Darstellung generiert werden muss. Dieser Prozess beginnt mit dem Auslesen der

[3] http://www.json.org/javadoc/org/json/JSONArray.html (aufgerufen am 07.05.2012

[4] bei 10 Seiten kann es bei einem Rechner mit 4 GB Arbeitsspeicher und einem 1,65 GHz Single Core Prozessor circa eine Minute dauern – abhängig natürlich von der Anzahl der snapshots innerhalb der einzelnen Wikiseiten

aktuellen WikiPage (Attribute, type tags etc.) sowie sämtlicher AttributeDefinition sowie TypeTagDefinition. Im Anschluss daran werden die einzelnen Änderungen an der WikiPage rekonstruiert und gleichzeitig die AttributeDefinition sowie TypeTagDefinition bis zum jeweiligen Zeitpunkt des aktuellen WikiPage-snapshots wiederhergestellt. Dieser Prozess allein ist sehr zeitintensiv und die darauf folgenden XPath-Abfragen, die auf einem relativ großen String operieren, können auch sehr viel Zeit in Anspruch nehmen. Aus diesem Grund werden zahlreiche verschiedene Caches eingesetzt.

XML-Partial-Caching Bei Erstellen der XML-Repräsentation auf eine Suchanfrage oder eine WikiPage werden sogenannte Partials verwendet. Der Begriff Partial wird in Ruby on Rails verwendet, um ein Template zu bezeichnen, das einen kleinen Abschnitt Code generiert und das nur einen Teil der letztlichen Gesamtausgabe darstellt und hat deshalb zur Verwendung des Begriffs in diesem Zusammenhang inspiriert.
Partials im Kontext dieser Bachelorarbeit sind XML-Quelltextfragmente, die das XML-Element <wikipage> beschreiben. Diesen Partials fehlt aber das einleitende root-Element, um eine gültige XML-Datei zu generieren. Aufgrund des fehlenden XML-root-Elements (in dieser Bachelorarbeit wurde das root-Element <usageAnalysis> gewählt) und auch wegen des dadurch veränderten absoluten Pfades, können auf Partials keine XPath-Abfragen sinnvoll ausgeführt werden. So würde etwa die Abfrage
`/usageAnalysis/wikiPage[1]/ attribute/key[1]`
kein Ergebnis zurückgeben, da die „richtige“ Abfrage
`/wikiPage[1]/attribute/key[1]`
heißen müsste. Einmal erstellte Partials werden gecacht (entweder im Dateisystem oder im Speicher, je nach der gewählten Caching-Klasse), so dass die snapshot-Generierung nicht mehrfach wiederholt werden muss. Standardmäßig wird der Memory-Cache verwendet. Um den gewählten Cache leicht auswechseln zu können, also leicht zwischen dem Dateisystem-Cache und dem Memory-Cache wechseln zu können, wird das Factory design pattern verwendet. Dabei wird angenommen, dass die WikiPage seit Erstellen des Partials nicht verändert wurde. Diese Annahme würde bei einem reinen DMKD-Projekt zutreffen, ist aber bei Tricia nicht notwendigerweise gegeben. Aus diesem Grund steht die Möglichkeit zur Verfügung, den Cache zu leeren und so eine Neuerstellung der snapshots zu erzwingen. Die Gültigkeitsdauer der im Dateisystem erstellten Partial-Dateien ist unbegrenzt, da dieser Cache vor allem zu Entwicklungszwecken gedacht ist, um veränderte Features im Programm schneller testen zu können. Anders verhält es sich mit der Gültigkeitsdauer der im Speicher gecachten Partials. Dieser Cache wird zum einen dann erneuert, wenn Tricia neu gestartet wird, zum Anderen wenn die Action Clear cache aufgerufen wird.

XML-Darstellung-Caching XML-Repräsentationen werden jeweils auf Anfrage generiert. Diese Anfrage kann entweder eine Suchanfrage (also ein serialisierter JSON-String) oder eine WikiPage mit deren UID sein. Es gibt also für jede erstellte XML-Darstellung einen String, der die jeweilige XML-Datei generiert. Aus diesem Grund wurde der Cache als key-value-Cache implementiert, der auf einen übergebenen String einen XML-String zurückgibt. Da sich eine UID (bestehend aus „wikipage/IDENTIFER“) und ein JSON-String (beginnend mit { und endend mit }) ist die Eindeutigkeit gegeben. Um nach zahlreichen An-

fragen nicht out of memory-Probleme zu bekommen, hat der Cache eine begrenzte Größe, die willkürlich auf 10 festgelegt worden ist. Aus Gründen der Einfachheit wird der Cache beim Überschreiten der maximalen Größe komplett geleert, um wieder Platz zu schaffen.

Caching komplexer XPath-Abfragen Mit Hilfe meines Plugins können komplexe XPath-Abfragen ausgeführt werden, die der Benutzer auch selbst definieren kann, indem die jeweiligen Tricia-Script-Funktionen in eine WikiPage eingebettet werden. Diese Abfragen können mittels nested loops den XML-Baum beliebig traversieren und dabei Daten in eine tabellenartige Datenstruktur schreiben. Da diese Abfragen auch von hoher Komplexität sein können und beim Neuladen der Seite auch wieder schnell dargestellt werden sollten, wird auch für diese Ergebnis-Datenstrukturen Caching angewandt. Ähnlich wie bei dem XML-Darstellung-Caching ist auch das Argument der Methode, die die tabellenartige Struktur erstellt, ein JSON-String. So wurde auch in diesem Fall ein key-value-Cache angelegt, der beim Erreichen der Maximalgröße geleert wird. Die Maximalgröße wurde auch in diesem Fall willkürlich auf 10 Elemente festgelegt, was aber auch durch die Annahme motiviert war, dass im Schnitt 10 solcher nested XPath-Abfragen in einer Seite dargestellt werden sollen, so dass beim Neuladen der Seite (etwa, weil man ein weiteres Diagramm eingebunden hat) man mit Cache-hits rechnen kann und so eine relativ zügige Ladezeit erreicht. Die Annahme, dass circa 10 nested XPath-Abfragen pro Seite verwendet werden, beruht auf eigenen Erfahrungen im Rahmen der von mir erstellen Dashboard.

5.13 Synchronisierte XPath-Abfragen

Die Verwendung des Cache stellt besondere Anforderungen an die Programmführung. Ein Dashboard kann mehrere XPath-Abfragen zu einem im worst case aufwendig zu generierenden XML-Dokument sein. Man stelle sich beispielsweise eine Abfrage über mehrere tausend WikiPages vor, die mehrere Stunden dauern kann zu der zahlreiche Visualisierungsgrafiken zu erstellen sind. Es kommt aber nur dann zu einem cache hit, wenn die XML-Generierung bereits abgeschlossen ist. Wenn nun zeitgleich zu einer XML-Datei (gleicher Suchstring oder gleiche UID) mehrere Anfragen eintreffen, käme es zu einem cache miss, da das XML-Dokument noch nicht fertiggestellt wurde und die XML-Datei würde ein weiteres Mal neu erstellt. Ohne besondere Vorkehrungen für jede einzelne Visualisierung das XML-Dokument neu generiert werden muss, da die Anfragen zeitgleich gestellt werden was zu einer Vervielfachung der Laufzeit führen würde: Für 10 Grafiken würde das etwa die zehnfache Laufzeit bedeuten. Aufgrund der ohnehin großen Laufzeit bei einer großen Zahl von Wikiseiten ist dieser Zustand untragbar. Das soeben skizzierte Problem ist zu vermeiden, indem die Cache-Klasse solange blockiert, bis das XML-Dokument fertiggestellt wurde. Das bedeutet allerdings auch, dass zu jeder Zeit nur ein XML-Dokument generiert und nur ein gecachtes XML-Dokument ausgegeben werden kann. Während also eine komplexe XPath-Abfrage ausgeführt wird, blockiert also der Cache, so dass neue XPath-Abfragen erst dann beantwortet werden, wenn die vorige Abfrage fertiggestellt ist. Durch diesen Blockierungsmechanismus wird dafür gesorgt, dass die XML-Generierung sequentiell erfolgen muss. Als Nebeneffekt kann auf diese Weise eine Blockierung des Systems durch eine große Zahl von Abfragen vermieden werden, da jeweils nur ein Thread an der XML-Generierung arbeiten kann.

5.14 Typdefinitionen und Attributdefinitionen

Typdefinitionen und Attributdefinitionen implementieren in Tricia das Versionable interface, so dass Änderungen an diesen Objekten dokumentiert werden. Es standen für die Extraktion dieser Definitionen keine Klassenmethoden zur Verfügung, so dass ich eine eigene Datenbankabfrage dafür geschrieben habe. Da es für die Rekonstruktion der Attributdefinitionen und Typdefinitionen nicht reicht, die Changes an aktuell verwendeten Attributnamen und type tags wiederherzustellen, habe ich jeweils eigene Methoden geschrieben, um alle aktuellen Attributdefinitionen bzw. Typdefinitionen auszulesen.
Der Grund dafür, dass sämtliche Definitionen (im Bezug auf Attributdefinitionen bzw. Typdefinitionen) rekonstruiert werden müssen, ist, dass es in der Vergangenheit Attributnamen mit zugehöriger Definition existiert haben können, wobei später entweder Attributname oder die AttributeDefinition gelöscht wurde und so im aktuellen Zustand nicht mehr vorhanden ist. Um den Zustand der Definition mit der untersuchten WikiPage zu synchronisieren, wird der Methode, die die Änderungen rückgängig macht ein timestamp übergeben, bis zu dem die Changes mittels `undo()` durchlaufen und revidiert werden sollen. Sobald dieser Zeitpunkt erreicht wird, wird die aktuelle Definition zurückgegeben (als XML) und merkt sich die aktuelle ChangeSet-Position. Diese Methode kann dann wieder aufgerufen werden (mit einem zeitlich früher gelegenen timestamp) um einen noch früheren Zustand der Definition zu rekonstruieren.
Es gibt allerdings keine „Geschichte“ einzelner Attribut- oder Typ-Definitionen. Das liegt daran, dass das XML-Dokument aus der Perspektive einer Wikiseite erstellt wird. Das bedeutet beispielsweise, dass wenn eine Attributdefinition einmal erstellt wurde, allerdings der entsprechende Attributname nie verwendet und anschließend die Attributdefinition gelöscht wurde, in keiner der von mir erstellten XML-Dokumente ein Hinweis auf die Existenz der Attributdefinition existiert. Selbiges gilt auch für Typdefinitionen. Eine solche Geschichte der Definitionen hätte zwar ohne Probleme in relativ kurzer Zeit implementiert werden können, allerdings ist das aus Zeitmangel nicht geschehen, zumal es lediglich eine andere Perspektive geboten hätte und Abfragen diesbezüglich nicht in den Forschungsfragen enthalten waren. Immerhin wird die Geschichte aller Attribut- und Typ-Definitionen, die in Wikiseiten bereits verwendet wurden, bereits vollständig abgedeckt.

Suche Tricia verfügt über ein mächtiges Suchwerkzeug, mit dem man Wikiseiten, Wiki, type tags, Attribute (und zahlreiche andere Assets, die für die Analyse im Rahmen dieser Bachelorarbeit allerdings nicht von Bedeutung sind) auswählen kann. Dabei kann spezifiziert werden, welcher Text im Volltext enthalten sein muss (query), welche type tags die Suchergebnisse haben müssen (must), welche Art von Assets zurückgegeben werden sollen (kind), also etwa Wikiseiten, Wikis etc. und aus welchem Wiki bzw. welchen Wikis die Ergebnisse stammen müssen (spaces).

Die Suchanfrage wird in einen JSON-String serialisiert, der an andere Handler weitergegeben werden kann. Aufgrund der Übersichtlichkeit der Suche mit den zahlreichen Einstellmöglichkeiten in der Sidebar habe ich mich entschieden, die Suchfunktion für die Selektion der zu untersuchenden Wikiseiten zu verwenden. Da man nur Wikiseiten analysieren können soll, wird die Action Analyse wiki page(s) nur dann angezeigt, wenn als kind `wikiPage` ausgewählt wurde, da meine Analyse jeweils Wikiseiten-basiert ist.

Im Suchergebnis können also eine oder mehrere Wikiseiten sein. Je nachdem ob es sich um eine Wikiseite oder um mehrere Wikiseiten handelt, wird jeweils ein anderes Dashboard dargestellt.

5.15 Snapshot-Generierung

Nachdem nun die wichtigen Konzepte des Plugins sowie Designentscheidungen erläutert wurden, werde ich im Folgenden noch die Snapshot-Generierung behandeln und dabei insbesondere die Verantwortlichkeiten der jeweiligen Klassen beschreiben.
Es gibt drei Arten von snapshots, die generiert werden müssen, um das XML-Dokument

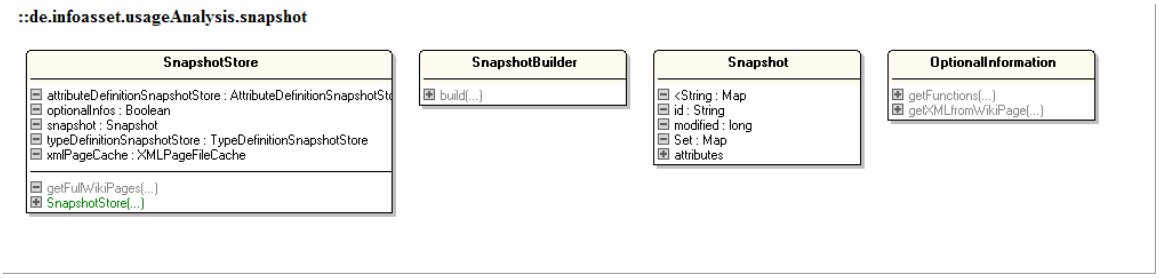


Abbildung 5.2: UML-Klassendiagramm für das snapshot-package

zu generieren:

- Wiki page-snapshots,
- Attributdefinition-snapshots sowie
- Typdefinition-snapshots.

Die Generierung der einzelnen snapshots läuft dabei jedes Mal nach einem ähnlichen Schema ab. Die `SnapshotStore`-Klasse stellt Methoden zur Verfügung, die die zu analysierenden Wikiseiten (für reguläre snapshots) bzw. den jeweiligen Wikispace (für Attribut- und Typdefinitionen) übernehmen und daraus ein XML-Dokument generiert und zurückgibt. Daraufhin wird der aktuelle Zustand mit Hilfe der statischen Methode `build()` rekonstruiert, die innerhalb der Klasse SnapshotBuilder befindlich ist. Anhand des aktuellen Zustands werden nun die Änderungen innerhalb der Klasse SnapshotStore rückgängig gemacht, wobei die Änderungen jeweils innerhalb der Snapshot-Objekte gespeichert werden.
Im Fall des WikiPage-SnapshotStores wird zur XML-Generierung die XML-Ausgabe der Snapshot-Objekte mit Wikiseiten-spezifischen XML-Ausgaben der SnapshotStore-Klasse ergänzt, wobei die Methoden `toXML()` und `toDiffXML()` eine XML-Repräsentation des Snapshots bzw. der Änderungen zwischen dem jetzigen und dem vorigen snapshot zurückgeben.
Die einzelnen Snapshots sind eine Hierarchiestufe unter den Wikiseiten angesiedelt, das heißt die snapshot-XML-Elemente sind Kindelemente von Wikiseiten. Die Aufgabenteilung besteht nun darin, dass die SnapshotStore-Klasse wikiseitenspezifische Informationen (z. B. Länge des Volltextes) der XML-Datei hinzufügt, während die Snapshot-Klasse

allein für die Generierung der snapshot-XML-Elemente verantwortlich ist.
Während die XML-Datei von Typdefinitionen bzw. Attributdefinitionen einzeln ablaufen können, so wird beim Rekonstruieren von Wikiseiten auch jedes mal der Zustand der Attribut- bzw. Typdefinitionen rekonstruiert. Aus diesem Grund enthält die SnapshotStore-Instanz einer Referenz auf den SnapshotStore von Attribut- und Typdefinitionen, um über den Aufruf der Methode `applyChanges(long until)` den Zustand von Attribut- bzw. Typdefinition auf den Zustand zu dem Zeitpunkt zurückzusetzen, der mittels `long until` übergeben wurde.

5.16 Zugriffsrechte

Zugriffsrechte sind auch bei der Implementierung meines Plugins von Bedeutung, da Wikiseiten und ihre Versionshistorie geladen werden, die jeweils Zugriffsberechtigungen unterliegen. Um nicht-autorisierten Benutzergruppen den Zugriff auf das Plugin zu verwehren, werden als Voraussetzung für den Zugriff schreibende Zugriffsrechte vorausgesetzt. Das Plugin verwendet dabei die Tricia-eigenen Methoden zur Zugriffsverwaltung, so dass nur Entities analysiert werden können, zu denen der Nutzer auch Zugriffsrechte besitzt. Allerdings könnten mit der Methode `fun()` die Zugriffsrechte „ausgehebelt“ werden, was zu einem „Datenleck“ und so zu einer Sicherheitslücke führen. Das ist nicht wünschenswert, so dass stattdessen nur Wikiseiten selektiert werden, die auch tatsächlich für den Benutzer lesbar sind.

6 Implementierung

6.1 SQL-basierter Prototyp

Zu Beginn der Bachelorarbeit wurde ein Prototyp in Java angefertigt, der aus einer MySQL-Datenbank eine XML-Datei generiert, die sämtliche WikiPages mit snapshots enthält und im Anschluss basierend auf dieser XML-Datei eine Anzahl von XPath-Queries ausführt. Zur Vorbereitung der Implementierung wurde MySQL installiert und eine Datenbank angelegt, die mit einem anonymisierten Database-Dump der Sebis-Website „befüllt" wurde. Um von den Daten in der Datenbank zu den Ergebnissen der XPath-Abfrage zu kommen, werden durch den Prototyp folgende Schritte ausgeführt:

1. Zunächst werden alle WikiPages aus der Tabelle wikipage_ ausgelesen,
2. dann werden von jeder WikiPage die Felder uhybridproperties sowie userializedtags als JSON interpretiert und daraus die entsprechende Datenstruktur befüllt.
3. Daraufhin werden zu jeweiligen WikiPage die Veränderungen aus der Tabelle changeset_ ausgelesen. In dieser Tabelle werden allerdings nicht nur Änderungen an den Attributen und type tags gespeichert, sondern unter anderem auch Änderungen am rich text. Würde jede einzelne row (also Ergebniszeile der SQL-Abfrage) zu einem neuen snapshot führen, gäbe es viele snapshot-Duplikate die sich nicht vom vorhergehenden snapshot unterscheiden. Das würde zum einen die Lesbarkeit des XML-Quelltextes verschlechtern als auch die XPath-Abfragen verlangsamen, da vor allem reine rich text Änderungen häufig vorkommen.
4. Bei relevanten Änderungen wird die entsprechende Datenstruktur verändert sowie eine flag gesetzt, dass ein neuer snapshot erstellt werden soll.
5. Bei gesetzter flag wird ein snapshot in die XML-Datei geschrieben und dabei wird eine Kopie der Datenstruktur angelegt. Mit dieser Kopie können dann die Änderungen zwischen zwei snapshots rekonstruiert werden und dann auch in die XML-Datei geschrieben werden.
6. Nachdem die XML-Datei generiert wurde, werden darauf verschiedene, beliebige XPath-Abfragen ausgeführt und in der Konsole ausgegeben.

Der Vorteil der SQL-basierten Analyse ist der hohe Optimierungsgrad und die daraus resultierende relativ geringere Laufzeit. Das liegt daran, dass aus den Queries direkt die jeweiligen Tabellenfelder ausgelesen werden und ein Umweg über Objektinstanziierung vermieden wird. Eine weitere Optimierung ist, dass die erstellten XML-Zeilen direkt in die Datei geschrieben werden, ohne dass sie zwischengespeichert werden. Das bedeutet, dass sobald eine WikiPage fertiggestellt wurde, von dieser kein Arbeitsspeicher mehr benötigt

wird, so dass sie aus dem Arbeitsspeicher gelöscht werden kann.
Die XPath-Abfrage, die im Anschluss erfolgt, kann allerdings bei einer größeren Dateigröße mehrere Minuten bis Stunden dauern. Das liegt daran, dass die XPath-Abfragen auf einem String ausgeführt werden. Vermutlich gäbe es eine bessere Performance, wenn die XML-Daten in eine spezielle XML-native Datenbank gespeichert werden, die für XPath-Abfragen optimiert ist.
Nachteil der direkten Abfrage per SQL ist allerdings, dass Addon-Funktionalität, die bei einer späteren Analyse nützlich sein könnte, schwer zu bewerkstelligen ist.
Diese SQL-basierte Vorgehensweise wurde zwar später nicht übernommen, half aber für ein grundlegendes Verständnis über die zugrundeliegende Datenbank. Die getter und setter-Methoden, die von Tricia sowie dem Hybrid Wiki-Plugin zur Verfügung gestellt werden, abstrahieren die die zugrundeliegenden Daten, so dass man nicht trivialerweise sehen kann, welche Daten zum jeweiligen Objekt in der Datenbank gespeichert werden – zumal zum Zeitpunkt der Entwicklung des Plugins keine Dokumentation für diese Aspekte des Quelltexts existierten. So ist zum Beispiel nicht ersichtlich, dass im ChangeSet der Handler gespeichert wird, der für die Veränderung verantwortlich ist. Dieses Wissen ist zum Beispiel hilfreich, um herauszufinden, ob ein spezieller Handler verwendet wurde, etwa der ExcelImportHandler.
Beim später adoptierten Klassen-basierten Verfahren wurden die Erkenntnisse aus diesem Prototyp eingeflochten.

1. Auf die direkte Abfrage per SQL wurde zugunsten einer Klassen-basierten Lösung verzichtet.
2. Die XML-Datei wurde um zahlreiche Zusatzinformationen erweitert, z. B. die Tricia-Script-Funktionen, die parent WikiPage sowie die child WikiPages
3. Die AttributeDefinitions sowie TypeTagDefinitions mussten für jeden snapshot rekonstruiert werden.

6.2 Laufzeit

Abfragen über einem ganzen Wikispace sind zeitaufwändig, da sämtliche ChangeSets zu einer Wikiseite durchlaufen werden und dabei die XML-Datei vergrößern, die in den Speicher geladen werden muss. Die Generierung der XML-Dateien und der Visualisierungen für die 323 Wikiseiten des Wikispace sebis Public Website dauert beispielsweise auf einem Rechner mit 4 GB Arbeitsspeicher und einem 1,65 GHz Single Core Prozessor gut 14 Minuten, wobei ein Großteil der Zeit für XPath-Abfragen benötigt wurde.
Bei gleichzeitigem Zugriff mehrerer Benutzer kann es deshalb zu sehr großen Antwortzeiten kommen die ggf. mit einem Timeout enden. Aus diesem Grund sollten nur autorisierte Benutzer auf diese Daten zugreifen dürfen. Plugintechnisch wurde dies mit der Forderung verwirklicht, dass Anwender auf die jeweiligen WikiPages nicht nur lesenden sondern auch schreibenden Zugriff haben müssen. Somit kann sichergestellt werden, dass lediglich Administratoren oder Personen mit ähnlichen Rechten bzw. Superrechten das Plugin verwenden können. Ohne entsprechend hohe Anforderungen an die Berechtigungen könnten bösartige Angreifer das System lahmlegen, indem sie es mit mehreren entsprechend umfangreichen Anfragen „bombardieren“.

Um die Laufzeit für wiederkehrende Anfragen zu verringern, wurden verschiedene Caches eingesetzt. Dieses Caches speichern (wie bereits unter Design erläutert):

1. spezielle XML-Dateien entweder im Dateisystem oder auf Wunsch im Speicher,
2. einzelne WikiPage-Repräsentationen im XML-Format, die als Bausteine für XML-Dateien verwendet werden sowie
3. komplexe XPath-Abfragen, die als Tabelle gespeichert werden.

Aufgrund des Anteils der Laufzeit von XPath-Abfragen an der gesamten Laufzeit ist vor allem Fall (3) beschleunigungstechnisch relevant, da damit die Zahl der XPath-Abfragen reduziert wird. Bezüglich der Laufzeit ist auch der XPath-Query-Evaluator von Bedeutung. Nach [GKPW03] steigt die Rechenzeit bei vielen XPath-Evaluatoren exponentiell mit der Größe der Eingabe. Es gebe aber bereits XPath-Evaluatoren, die in Polynomialzeit zu einem Ergebnis kommen. Aktuell wird allerdings das javax.xpath-packages für die XPath-Funktionalität verwendet.

6.3 Testverfahren

Um die Korrektheit der Ergebnisse zu gewährleisten, muss primär das generierte XML-Dokument korrekt sein. Aus diesem Grund habe ich anhand der bereits in Tricia integrierten Versionshistorie überprüft, welche Änderungen zu welchem Zeitpunkt vorgenommen wurden und somit das XML-Dokument rekonstruiert und mit dem Plugin-generierten XML-Dokument verglichen.
Da die XPath-Abfragen mittels der Javax-Bibliothek ausgeführt wurden, also dem Tool eines Fremdanbieters, kann davon ausgegangen werden, dass die XPath-Abfragen korrekt sind und bereits ausgiebig entsprechend der Spezifikation durch das W3C-Konsortium getestet wurden.

7 Bedienung und Verwendung des Plugins

Installation Die Integration erfolgt durch Eintragung des Plugins in die Tricia-Konfigurationsdatei, wobei beachtet werden muss, dass dieses Plugin vom Plugin HybridWiki abhängt.

Dashboards Wie bereits an anderer Stelle erwähnt, stehen Standardvisualisierungen bei einzelnen Wikiseiten mit der View „Analyze wiki pages“ zur Verfügung; um den Cache zu löschen und aktuelle Ergebnisse zu erhalten, steht die Action „Clear cache“ bereit. Bei Wikis steht die View „Analyze definitions“ zur Verfügung, um Typ- und Attributdefinitionen eines Wikis analysieren zu können.

eigene Visualisierungen Es stehen drei Tricia-Skript-Funktionen zur Verfügung, um eigene Abfragen oder Visualisierungen in Wikiseiten einzubetten:

- xpath für einfache XPath-Abfragen, wobei es sich um eine print substitution handelt,
- list in Kombination mit xpathResults für komplexe Abfragen, wobei es sich um eine list substitution handelt,
- xpathVisualization für Visualisierungen der beiden obigen Abfragen mit Hilfe der Google Charts API, wobei es sich wiederum um eine print substitution handelt.

xpath Am einfachsten zu verwenden ist die Tricia-Funktion `xpath`. Diese Funktion hat die Parameter

- `xpath` (obligatorisch), der die XPath-Abfrage als String angibt
- `xpathDiv` (optional[1]), um gegebenenfalls eine weitere XPath-Abfrage zu erstellen, wobei der Quotient von `xpath` und `xpathDiv` gebildet und ausgegeben wird. Bei einer ungültigen Division (z. B. Division durch 0) wird -1 ausgegeben.
  Dieser Funktionsparameter ist allerdings redundant, da auch die XPath-eigene `div`-Funktion verwendet werden kann und sollte deshalb nur verwendet werden, falls im Fehlerfall -1 zurückgegeben werden soll.
- `uid` (optional) gibt die UID der Wikiseite an, die analysiert werden soll. Die UID einer Wikiseite beginnt jeweils mit „wikipage/“, z. B. wikiPage/1xy6w6pb8rf9j
- `search` (optional) enthält einen serialisierten Suchstring, der im JSON-Format kodiert ist. Der Suchstring enthält als `kind` verpflichtend nur das Element `wikipage` und wird von der Tricia-eigenen Suchfunktion generiert.

[1] redundant

- Werden sowohl `uid` als auch der Parameter `search` übergeben, so wird `search` bevorzugt behandelt.
- Ist der optionale Parameter `typetagdefinitions` gesetzt, so wird keine Wikiseite analysiert, sondern die Typdefinitionen. Enthält der Parameter `typetagdefinitions` den Wert „default“, so werden die Typdefinitionen des Wikis analysiert, in dem sich die Wikiseite befindet, die die Tricia-Script-Funktion definiert. Andernfalls kann als Parameter die UID des Wiki-UID übergeben werden (oder – falls nur die Wikipage-UID zur Verfügung steht auch diese), um die Analyse auf den Typdefinitionen des jeweiligen Wikis durchzuführen.
- Analog zu `typetagdefinitions` gibt es den Funktionsparameter `attributedefinitions`, der ebenfalls entweder den Wert default oder die UID eines Wikis enthalten kann.
- Fehlen die Parameter `uid`, `search`, `typetagdefinitions` und `attributedefinitions`, wird versucht, den search- bzw. uid-Parameter aus dem Handler zu entnehmen.

Beispiel 7.1 *$xpath(xpath="count(/*/wikipage/snapshot/attribute[count(./value)>1]) div count(/*/wikipage/snapshot/attribute)")$*
Diese XPath-Abfrage berechnet die Anzahl der Attribute mit mehr als einem Wert und teilt das Ergebnis durch die Anzahl aller Attribute. Zu beachten bei der Interpretation dieses Ergebnisses ist, dass bei dieser Abfrage alle Attribute in allen snapshots aller Wikiseiten herangezogen werden.

Beispiel 7.2 *$xpath(xpath="count(/*/wikipage/snapshot[1]/type)")$*
Diese XPath-Abfrage berechnet die Anzahl aller Typen in allen aktuellen Wikipage-snapshots, beantwortet also die Frage: Wie viele Typen werden derzeit verwendet?

Für weitere Beispiele zu dieser Tricia-Script-Funktion kann auch das Dashboard zu Hilfe genommen werden: bewegt man den Mauszeiger über einen Wert, der mittels einer Abfrage generiert wurde, so erscheint ein Feld, in dem die Tricia-Script-Funktion mit der zugehörigen XPath-Abfrage erscheint, die den jeweiligen Wert erzeugt hat.

list Die soeben vorgestellte Funktion hat eine entscheidende Limitierung: es können nur einzelne Werte ausgelesen werden. Die Methode xpathResults gibt eine Liste von Schlüssel-Wert-Paaren aus, die mit Hilfe der Tricia-Script-Funktion $list()$ in eine list substitution verwandelt wird. Diese Funktion hat die Parameter

- `name` (obligatorisch) muss „xpathResults“ sein.
- `uid`, `search`, `typedefinitions` und `attributedefinitions` stehen auch bei dieser Funktion zur Verfügung; die Verwendung erfolgt analog zu der Erklärung, die zuvor für die Funktion $xpath()$ gegeben wurde.
- `controls` (optional) übergibt Informationen an die Funktion „xpathResults“, die angeben, wie die XPath-Abfrage ausgeführt werden soll.
  Sofern nur eindeutige Werte gewünscht sind, kann hier ein JSON mit `{unique:`

`true}` übergeben werden; als Nebeneffekt wird dann eine substitution angelegt, die die Häufigkeit der einzelnen eindeutigen Werte angibt (Name der substitution: frequency). Es sei an dieser Stelle darauf hingewiesen, dass es auch mit XPath spezifiziert werden kann, nur eindeutige Werte zu selektieren, allerdings geht das nicht mit dem grundlegenden, einfachen subset der XPath-Sprachelementen, die ich im Appendix unter XPath vorstellen werde. Dann steht allerdings nicht der key `frequency` zur Verfügung.

- `xpath` (obligatorisch) enthält eine JSON-Datenstruktur, die verschachtelte XPath-Abfragen ermöglicht. In der einfachsten Form handelt es sich dabei um eine Liste mit einer Schlüssel-Wert-Map, die die Schlüssel `xpath` und `values` enthält, die beide obligatorisch sind.
  Unter dem Schlüssel `xpath` wird die `XPath`-Abfrage gespeichert.
  `nested` (optional) kann dazu verwendet werden, um verschachtelte Unterabfragen festzulegen; `nested` ist dabei rekursiv definiert, kann also wiederum eine Liste von Maps enthalten mit xpath und values, so dass verschachtelte Abfragen in beliebiger Tiefe möglich sind. Die Werte, die extrahiert werden sollen sind unter dem Schlüssel `values` aufgelistet: `values` enthält eine Liste von Schlüssel-Wert-Maps, die folgende Schlüssel enthalten: `xpath`, `name` und `typ`.
  `xpath` gibt dabei wiederum die die XPath-Abfrage an, idealerweise relativ zum parent XPath-String. Zum Beispiel, wenn man zunächst
  `/*/wikipage/snapshot/attribute`
  ausgewählt hat, sollte man unter `values ./key` wählen.
  `name` enthält den Namen, der später als substitution zur Verfügung steht; interpretiert man das Ergebnis dieser Abfrage als Tabellenstruktur, so wäre `name` eine Spalte bzw. in SQL-Terminologie eine column.
  Soll die jeweilige column in JavaScript eingebettet werden, so sollte der jeweilige Name um das Suffix „_js“ erweitert werden. Zu jeder Spalte (mit Ausnahme von frequency, das bei eindeutigen Werten angelegt wird) wird automatisch eine spezielle Spalte für JavaScript angelegt, was den Vorteil hat, dass sofern nur die substitution mit dem Suffix „_js “ verwendet wird, kein Schadcode eingeschleust werden kann.
  `type` gibt den Typ an, der vor allem für JavaScript und die Google Charts API von Bedeutung sind. Die zur Verfügung stehenden Typen sind:
  - number
  - string
  - date

  Zu beachten ist die Kleinschreibung der Typen.

Zur Verdeutlichung des Funktionsparameters `xpath` soll die EBNF-Darstellung helfen:

```
ROOT            :=  „[“ XPATH_QUERIES „]“
XPATH_QUERIES   :=  XPATH_QUERY [ „,“ XPATH_QUERIES ]
XPATH_QUERY     :=  „{ xpath: ’“ XPATH_EXPRESSION „’,
                    values: [“VALUES „]“
                    [ „, nested: “ ROOT ]
                    “}“
VALUES          :=  „“ | „{ xpath: ’“ XPATH_EXPRESSION „’,
                    name: ’“ STRING_LITERAL „’,
                    type: ’“ TYPE „’ }“
TYPE            :=  „string“ | „number“ | „date“
```

Listing 7.1: Beispiel für die Verwendung der $list()$-Funktion

```
$[list(
        name="xpathResults",
        controls="{unique: true}",
        xpath="
        [{
                xpath: '/*/wikipage/snapshot[1]/attribute[./
                    attributeDefinition]',
                values: [{
                        xpath: './key',
                        name: 'key',
                        type: 'string'
                }]
        }]
        ") l$ [$l.key_js$,$l.frequency$], $list]$
])$
```

xpathVisualization Diese Funktion wurde so konzipiert, dass mittels der Funktionsparameter der JavaScript-Code für die Google Chart-Visualisierungen erstellt wird. In erster Linie soll die Funktion sicherstellen, dass kein Schadcode zur Ausführung gebracht werden kann. Um Benutzern, die die Google Chart-Visualisierungen kennen, die Arbeit zu erleichtern, orientiert sich die Namensgebung der einzelnen Tricia-Script-Funktionsparametern am entsprechenden JavaScript-Code. Ziel dieser Vorgehensweise ist es, dass ein Benutzer einen bestehenden Google-Chart-JavaScript-Code möglichst einfach in dessen Entsprechung in der Funktion xpathVisualization umwandeln kann und umgekehrt.

Listing 7.2 und 7.3 demonstrieren die Verwendung dieser Tricia-Script-Funktion.

Diese Funktion hat die Parameter

- `uid`, `search`, `typedefinitions` und `attributedefinitions` stehen auch bei dieser Funktion zur Verfügung; die Verwendung erfolgt analog zu der Erklärung, die zuvor für die Funktion $xpath()$ gegeben wurde.

- `visualization` (obligatorisch) ist ein JSONObject mit folgenden Schlüsseln:
  - `packages` (obligatorisch) stellt eine Liste von packages dar, die geladen werden sollen, kann aber auch leer sein sofern nur das Standard-packages geladen werden soll; die Schreibweise dafür lautet in JSON []
  - `columns` (obligatorisch) ist eine Liste einer Key-Value-Map die die Schlüssel type und name enthält. Unter type werden die columns für die Google Chart API definiert, während Name den Namen der Spalte enthält.
  - `rows` (obligatorisch) ist eine Liste von Zeilen, die jeweils eine Liste von Spalten enthält. Wenn dabei ein String übergeben wird und dieser String als Spaltenname in einer der XPathResult-Abfrage vorkommt, wird die jeweilige Spalte aus der XPathResult-Abfrage zurückgegeben; andernfalls wird der String escaped und ausgegeben.
  - `chart` (obligatorisch) ist die Google Charts API-Visualisierungsklasse. Mögliche Werte sind AnnotatedTimeLine, AreaChart, BarChart, BubbleChart, CandlestickChart, ChartEditor, ChartWrapper, ColumnChart, ComboChart, Gauge, GeoChart, LineChart, OrgChart, PieChart, ScatterChart, SteppedAreaChart, Table und TreeMap [2]. Zu beachten ist, dass manche der soeben genannten Visualisierungsklassen den Import von packages benötigen; weitere Informationen dazu gibt es in der Google Chart API.
  - `debug` (optional) ist ein Boolean-Wert, der – wenn auf true gesetzt – den generierten JavaScript-Quelltext als Text zurückgibt, um zu überprüfen, ob der JavaScript-Code korrekt ist.
- `xpath` (optional) ist erforderlich, falls xpathResults verwendet wird. Die Verwendung von xpath ist analog zur obigen xpathResults-Funktion.
- `controls` (optional) ist ebenfalls erforderlich, falls xpathResults verwendet wird. Die Verwendung von controls ist analog zur obigen xpathResults-Funktion.

Zu beachten ist, dass Tricia-Script mit Leerzeichen oder neuen Zeilen (aktuell) noch nicht umgehen kann. Aus diesem Grund muss alles in einer Zeile stehen und zwischen Funktionsparametern dürfen keine whitespace-Zeichen sein.

Listing 7.2: Beispiel für die Verwendung der $xpathVisualization()$-Funktion.

```
1 $xpathVisualization(visualization="{
2         columns:[
3                 {name: 'Typ', type: 'string'},
4                 {name:'Haeufigkeit', type: 'number'}
5         ],
6         draw: {title: 'aktuelle Haeufigkeiten von Attributtypen',
              width: 400, height: 400}, chart: 'BarChart',
7         rows:[['StringValue',{xpath: 'count(/*/wikipage/snapshot
             [1]/attribute[./value/@type=\'StringValue\'])'}]],
```

[2] vergleiche https://code.google.com/apis/ajax/playground/?type=visualization (aufgerufen am 08.05.2012)

```
        debug: false
}",uid="wikiPage/1xy6w6pb8rf9j")$
```

Listing 7.3: Weiteres Beispiel für die Verwendung der $xpathVisualization()$-Funktion.

```
$xpathVisualization(
        visualization="{
                packages: ['table'],
                columns:[
                        {name: 'Art', type:'string'},
                        {name: 'URL', type:'string'},
                        {name:'snapshot-Datum', type: 'date'},
                        {name:'bearbeitet', type: 'number'},
                        {name:'hinzugefuegt', type: 'number'},
                        {name:'entfernt', type: 'number'}],
                draw: {title: 'Snapshots - Aenderungen zwischen
                   Snapshots', width: 600, height: 600},
                chart: 'Table',
                rows:[['Wikiseite','url_js','when_js','
                   bearbeitet_js','hinzugefuegt_js','entfernt_js
                   ']],
                debug: false
        }",
        uid="wikiPage/1xy6w6pb8rf9j",
        xpath="[{
                xpath: '/*/wikipage',
                values: [{xpath: './@url', name: 'url', type: '
                   string'}],
                nested: [{
                        xpath: './snapshot',
                        values: [
                                {xpath: './@when_long', name: '
                                   when', type: 'date'},
                                {xpath: 'count(./diff/edited/*)',
                                    name: 'bearbeitet', type: '
                                   number'},
                                {xpath: 'count(./diff/added/*)',
                                   name: 'hinzugefuegt', type: '
                                   number'},
                                {xpath: 'count(./diff/removed/*)
                                   ',name: 'entfernt', type: '
                                   number'}
                        ]
                }]
        }]")$
```

8 Auswertung und Interpretation der Nutzungsanalyse

8.1 Verwendete Daten

Die verwendeten Nutzungsdaten sind die Daten der sebis Public Website, also der Website des Lehrstuhls für Software Engineering for Business Information Systems an der TU München. Die Daten wurden anonymisiert, um eine Zuordnung der Editoren auszuschließen und in Form eines SQL-Datenbankdumps zur Verfügung gestellt. Dabei stehen mehr als 50000 ChangeSet-Einträge zur Verfügung, wobei ChangeSets vom 27.06.2008 bis zum 23. November 2011 enthalten sind. Die Größe des Datenbank-Dumps beträgt ca. 380 MB und es dauerte auf meinem Rechner ca. 40 Minuten, um den Dump per Kommandozeile in die Datenbank zu importieren.
Zu beachten ist, dass die Daten von dem Lehrstuhl stammen, der Tricia selbst programmiert hat. Das bedeutet, dass die Strukturierungskonzepte und -mechanismen möglicherweise vorbildlicher genutzt wurden, als wenn Laien mit Tricia arbeiten. Insofern sind die Daten auch nur bedingt aussagefähig. Allerdings wurde im Rahmen dieser Bachelorarbeit ein generisches Tool entwickelt, das überall, wo Tricia eingesetzt wird, eingebunden werden kann und dann ohne weitere Konfiguration sofort Standard-Visualisierungen anzeigt. Aus diesem Grund ist die analysierte sebis Public Website lediglich als exemplarisch anzusehen.

8.2 Forschungsfragen

Für die Nutzungsanalyse kann mit Hypothesen gearbeitet werden oder ein explorativ-empirischer Ansatz verwendet werden. Beim ersteren Ansatz werden zu Beginn Hypothesen aufgestellt und diese Hypothesen im Anschluss auf Richtigkeit überprüft. Beim explorativ-empirischem Ansatz werden zunächst Daten gewonnen und aus diesen werden dann Schlüsse gezogen.
Bei dieser Bachelorarbeit wurden zu Beginn der Arbeit zahlreiche Hypothesen aufgestellt, die mit Hilfe eines Tricia-Plugins zu beurteilen waren:

q1: Wie werden die Strukturierungskonzepte und -mechanismen von Hybrid Wikis verwendet? Zur Beantwortung dieser Frage soll im Rahmen dieser Bachelorarbeit ein Tool entwickelt werden, das die Evolution der verwendeten Strukturierungskonzepte visualisiert. Auf die verwendeten Mechanismen kann zum Teil rückgeschlossen werden. Auf die durch die Auto-Vervollständigung vorgeschlagenen Typen, Attributnamen und -werte kann rückgeschlossen werden, da man die zum Zeitpunkt der Attribut- oder Typeingabe existierenden Attribute bzw. Typen kennt. Selbiges gilt für Attribut- und Typvorschläge.

Da die Handler-Klasse, die für die Attribut/Typ-Änderung verantwortlich ist, gespeichert wird, kann auch auf die Verwendung der jeweiligen Handler Rückschlüsse gezogen werden.

q2: Wie können die Messergebnisse verwendet werden, so dass die Nutzer von Hybrid Wikis davon profitieren? Diese Frage kann aufgrund der zugrundeliegenden Fragen im Fall der öffentlichen Website des sebis-Lehrstuhls noch nicht ausführlich beantwortet werden. Das liegt daran, dass der Lehrstuhl, der die Entwicklung der Hybrid Wikis betrieb, erwartungsgemäß die zur Verfügung stehenden Funktionen besser nutzt als jemand, der als Laie ohne oder mit nur kurzer Einführung Hybrid Wikis verwenden soll. Ich würde erwarten, dass Funktionen wie „Extract wiki pages“ kaum gefunden werden, weil sie schwer auffindbar sind. Es gab bereits eine Bachelorarbeit[Wie11], die sich mit dem Thema usability auseinandergesetzt hat. In dieser Bachelorarbeit wurde im Fazit vorgeschlagen, per Mouse-over bestimmte Funktionen zu erklären. Ich denke, dass es grundsätzlich hilfreich wäre, kontextabhängig Tipps zu geben, so etwa dass vor der Eingabe von Attributen darauf hingewiesen wird, dass die Attribute auch aus einer Tabelle importiert werden können. Auf diese Weise würde die Nutzung von Hybrid Wikis auch für Personengruppen leichter, die keinen so intuitiven Zugang zu Benutzeroberflächen haben und nach längerer Auszeit die Bedienung neu erlernen müssten.
Im Folgenden sollen zahlreiche Hypothesen überprüft werden.

H1: Hybrid Wiki Strukturierungskonzepte werden verwendet. Diese Hypothese kann einfach belegt werden, indem der prozentuale Anteil der Seiten, die Attribute bzw. type tags verwenden gemessen wird. Zu diesem Zweck kann das im Rahmen dieser Bachelorarbeit entwickelte Plugin entwickelt werden. Dazu werden zunächst in der Suche alle WikiPages aus dem gegebenen Wiki space ausgewählt und dann die Action „Analyze wiki pages“ gestartet. Dabei wird eine Reihe von XPath-Queries ausgeführt, unter anderem auch diese XPath-Abfragen:

- count(/*/wikipage[not(./snapshot[1]/attribute)])
- count(/*/wikipage[(./snapshot[1]/attribute)])
- count(/*/wikipage[not(./snapshot[1]/type)])
- count(/*/wikipage[(./snapshot[1]/type)])

Diese und viele weitere Abfragen sind in Abb. 8.3 dargestellt.

H2: Hybrid Wiki Strukturierungsmechanismen werden verwendet. Diese Hypothese lässt sich belegen, indem man zeigt, dass die neu erstellten Attributnamen und -werte sowie type tags bereits vorher verwendet wurden. Denn in diesem Fall kann die Vorschlagfunktion diese einblenden. Zu zeigen wäre also, dass die jeweiligen Strukturierungskonzepte mehrfach verwendet werden. Die Verwendung der Strukturierungsmechanismen

- extract wiki page (Attributwerte in Links umwandeln)
- apply4all-Mechanismus zur Konsolidierung von Integritätsbedingungen

- HTML2Hybrid bzw. Extract wiki pages

kann gezeigt werden, indem man zeigt, dass die jeweiligen Handler verwendet werden und die entsprechenden Attribute bzw. types angelegt werden. Das ist wie man an Abbildung 8.2 sieht, der Fall.

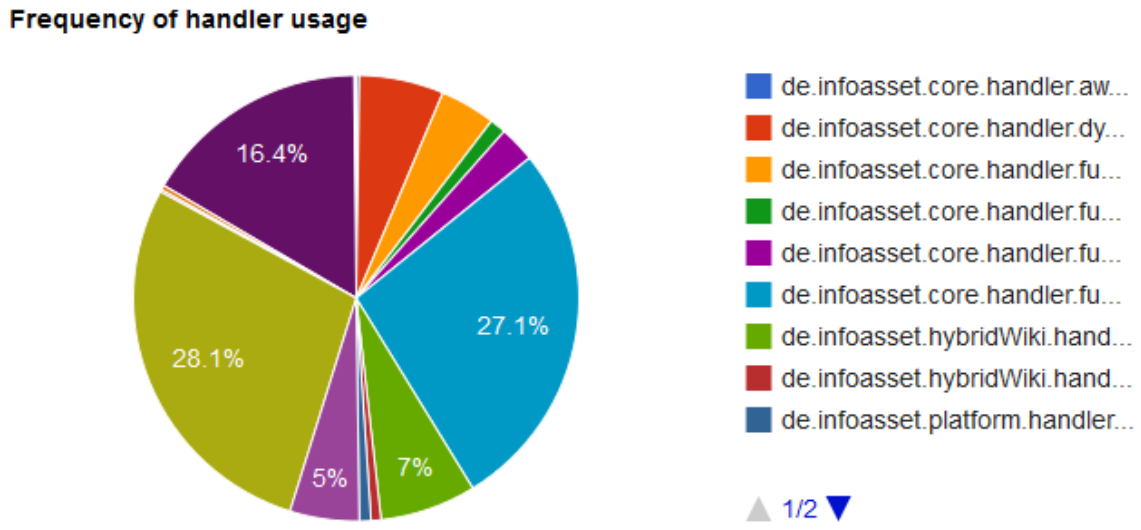


Abbildung 8.1: Diese Visualisierung zeigt, welche Handler an den Änderungen am strukturierten Teil des Wikis „sebis Public Website“ beteiligt waren. Mittels Mouseover wird der Handlername im Dashboard komplett angezeigt.

H3: Struktur entsteht inkrementell, d.h. der Strukturkern um Attribute und Relationen entsteht schrittweise über einen längeren Zeitraum hinweg und nicht innerhalb kurzer Zeit. Diese These lässt sich belegen, indem man sich die Attributentwicklung und typ-Entwicklung in einem Diagramm ansieht. Dabei werden auf der x-Achse die Zeit und auf der y-Achse die Attribute und type tags angetragen. Die Auswertung zahlreicher solcher Diagramme hat mir gezeigt, dass eine inkrementelle Entwicklung der Struktur üblich ist. Abb. 8.2 zeigt die Evolution der Wikiseite „Hybrid Wikis“.

H4.1: Hybrid Wiki Mechanismen führen zu einer Strukturkonvergenz, also zu einer einheitlicheren Verwendung der Struktur. Im Fall der Autovervollständigung bedeutet das, dass (a) Attributnamen und (b) -werte sowie (c) Typen aus der Autovervollständigung übernommen werden.

Eine Annahme in dieser Bachelorarbeit ist, dass die Übernahme eines von der Autovervollständigung vorgeschlagenen Strukturierungskonzeptes (zumindest zum Teil) das Ergebnis des Vorschlags ist. Um zu zeigen, dass das jeweilige Strukturierungskonzept aufgrund der Autovervollständigungsfunktion und nicht „ohnehin“ eingegeben wurde, müsste überprüft werden, welche Strukturierungskonzepte ohne der Autovervollständigungsfunktion gewählt worden wären. In einem Versuch könnten Testpersonen Attribute für eine WikiPage einmal mit und einmal ohne der Autovervollständigungsfunk-

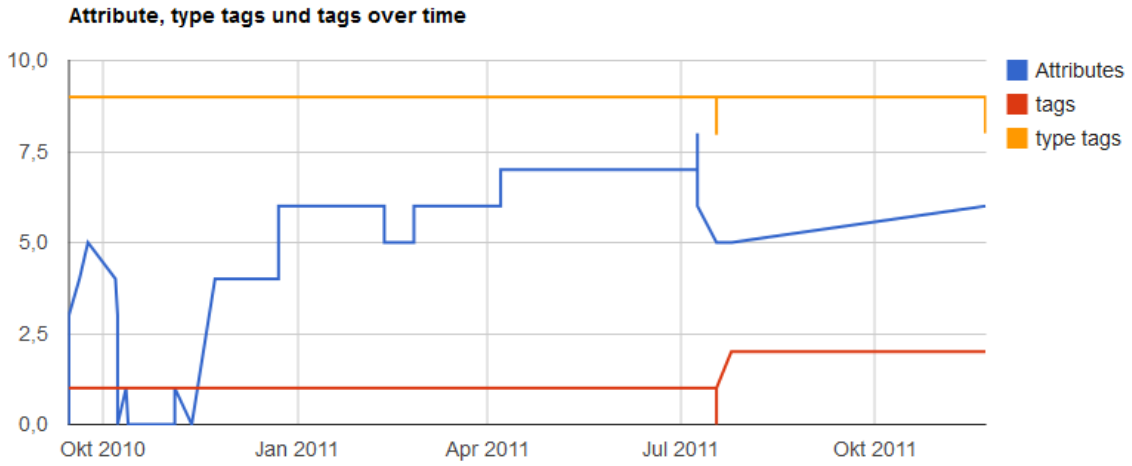


Abbildung 8.2: Diese Abbildung zeigt die Anzahl der Attribute, Typen und Tags der Wikiseite „Hybrid Wikis“ im Verlauf der Zeit beginnend mit dem ersten snapshot. Die Genauigkeit der Visualisierung beträgt einen Tag; wurde die Anzahl an einem Tag mehrfach geändert, erscheint ein senkrechter „Strich“, der über alle Werte geht, die an diesem Tag angenommen wurden.

tion eingeben. Wenn sich herausstellt, dass die Testpersonen ohne der Autovervollständigungsfunktion die Strukturierungskonzepte anders verwenden, würde das die Annahme bestätigen.
Abb. 8.2 zeigt die Häufigkeit von einzelnen Attributen, wobei die häufige Verwendung einzelner Attribute dafür spricht, dass die Autovervollständigung genutzt wird.
Die Vereinheitlichung von Attributnamen mittels apply4all kann allerdings bereits an der Verwendung des Handlers gezeigt werden.

H4.2: Konzepte „erhärten“ mit der Zeit, d.h. werden unmittelbar nach dem Entstehen stark verändert, nach längerem Bestehen hingegen nur noch wenig. Diese Hypothese kann wiederum belegt werden, indem gezeigt wird, dass Attributvorschläge und Typvorschläge übernommen werden. Auch hier müsste streng genommen mit Hilfe von Probanden gezeigt werden, dass die Vorschläge verwendet werden und die Benutzer nicht ohnehin die Absicht hatten, die entsprechenden Attribute und Typen anzulegen. Außerdem wäre zu zeigen, dass Seiten mit Typen zu denen es Attributvorschläge gibt, Attribute haben. Das kann gezeigt werden, indem man die einzelnen Typen den jeweiligen Wikiseiten mit ihren Attributen gegenüberstellt.

H5: die Strukturierungsmechanismen von Hybrid Wikis werden genutzt, um vom Unstrukturierten zum Strukturierten zu gelangen bzw. die Qualität/Quantität der Struktur zu erhöhen. Das Szenario besagt also allgemein, dass die Reduktion des Volltexts mit einer Vergrößerung des Strukturteils einhergeht. Dabei werden die Tricia-eigenen Strukturierungsmechanismen verwendet, um im Lauf der Zeit Struktur im Textbereich in Struktur mittels Strukturierungskonzepten zu überführen.

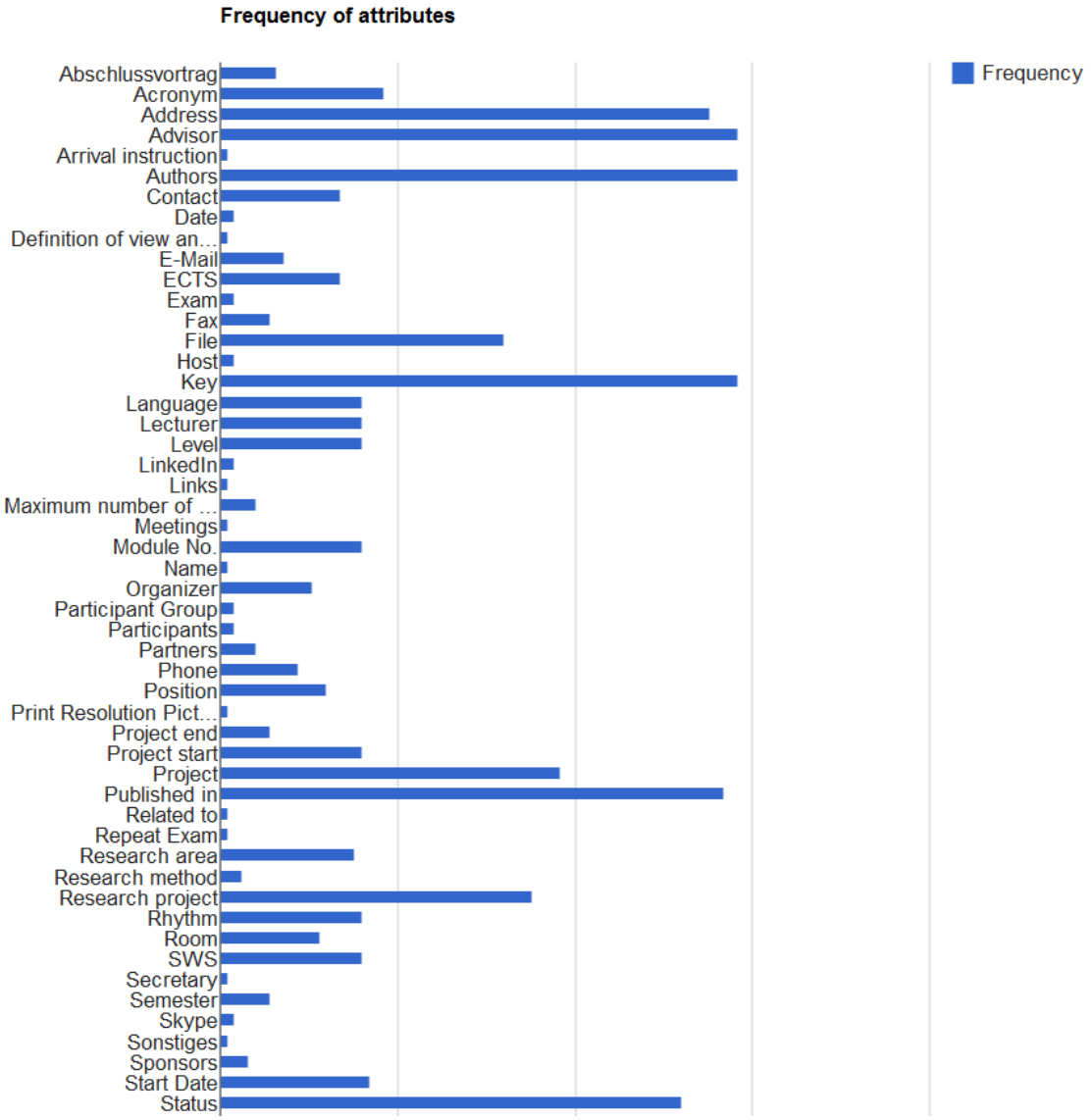


Abbildung 8.3: Häufigkeit von Attributen im des Wiki „sebis Public Website“

(z. B. Extraktionsmechanismen für Tabellen, genannt Extract Wiki Page, um Attributwerte in Links umzuwandeln). Sowie: Typen entstehen durch den Übergang von tags nach type tags, gegebenenfalls per drag and drop von Typen – eine Hypothese, die mit Hilfe der verwendeten Handler beurteilt werden kann.
Nach der Analyse verschiedener Seiten konnte ich wenige Seiten finden, deren Volltextlänge tatsächlich abgenommen hat, während der Strukturteil dementsprechend zugenommen hat. Das könnte daran liegen, dass Attribute Informationen nur steckbriefartig aufnehmen können; und wegen ihrer steckbriefartigen Natur können sie Struktur im Volltextbereich nicht ersetzten.
Für manche Informationen ist es allerdings nötig, die jeweils abgelegten Informationen zu kommentieren.

Beispiel: Klausurbeginn: 02.08.2011
Bitte mindestens eine halbe Stunde vor Klausurbeginn da sein!
Außerdem wurden Attribute im untersuchten Zeitraum nach Namen sortiert, während es manchmal sinnvoll ist, gewisse Attribute zu gruppieren (z. B. Adresse). Bis dato würde man Informationen, die zu gruppieren sind, in einer separaten Wikiseite anlegen, was allerdings nicht in allen Fällen wünschenswert ist. Beispielsweise könnten „Wichtige Hinweise zur Klausur“ statt auf der Vorlesungsseite in einer separaten Wikiseite angelegt werden; das kann allerdings dazu führen, dass diese Information übersehen wird. Ob die lineare Strukturierung von Attributen den Strukturierungsprozess erschwert, müsste allerdings in einer anderen Arbeit geklärt werden. Es ist anzumerken, dass gegen Ende der Bearbeitungszeit der Bachelorarbeit eine Gruppierungsfunktion anhand von Typen eingefügt wurde, so dass bei künftigen Nutzungsdaten eine Änderung im Nutzungsverhalten von Attributen und Typen denkbar wäre – allein um Attribute zu gruppieren.

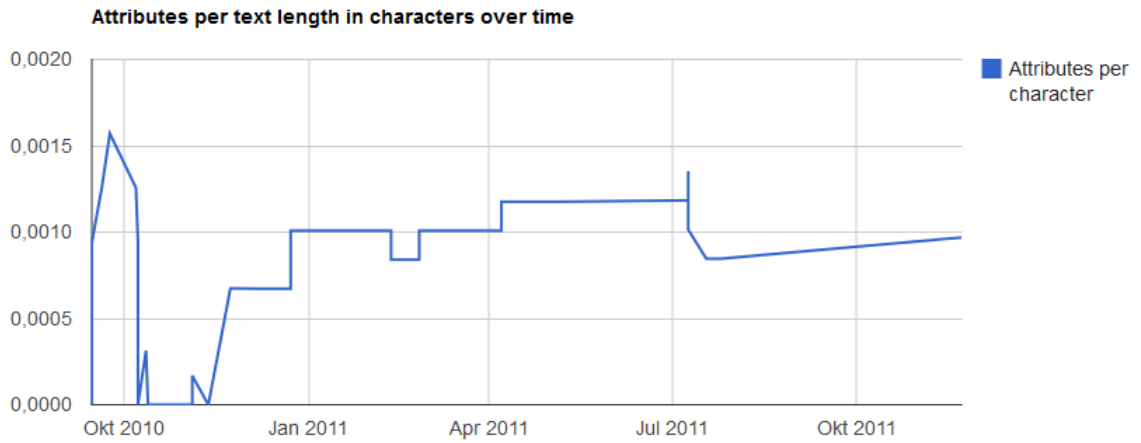


Abbildung 8.4: Diese Abbildung zeigt die Dashboard-Visualisierung für den Verlauf der Attribute pro Textzeichen der Wikiseite „Hybrid Wikis“. Zur Referenz existiert auch eine Grafik, die die Anzahl der Textzeichen im Zeitverlauf darstellt: Abb. 8.2

H6: Attribute und Typen werden unabhängig voneinander genutzt.
(a) Es gibt Seiten, die nur Attribute besitzen.
(b) Es gibt Seiten, die nur type tags besitzen.
Dies kann analog zu Hypothese H1 mit meinem Plugin gezeigt werden. Es handelt sich dabei um folgende XPath-Abfragen:

- (a) count(/*/wikipage[(./snapshot[1]/attribute) and not(./snapshot[1]/type)])
- (b) count(/*/wikipage[not(./snapshot[1]/attribute) and (./snapshot[1]/type)])

Die Auswertung ergab hier, dass Attribute und Typen meist in Kombination miteinander verwendet werden. Die Abfrageergebnisse sind in Abb. 8.3 dargestellt und sprechen tendenziell gegen eine unabhängige Nutzung von Attributen und Typen, stattdessen gehen Attribute mit Typen oft einher.

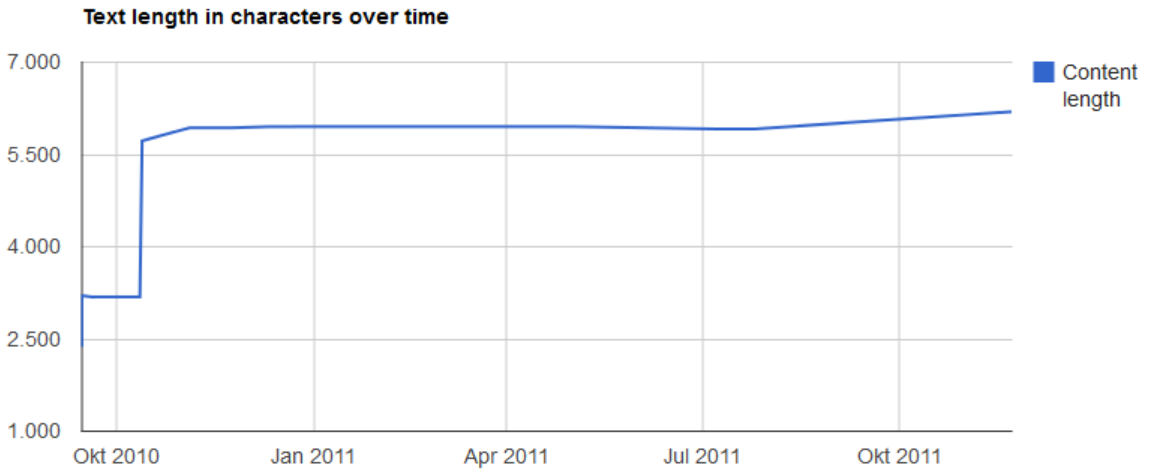


Abbildung 8.5: Diese Standard-Visualisierung zeigt die Textlänge der Wikiseite „Hybrid Wikis“ im Verlauf der Zeit und dient dabei als Referenz für die Visualisierung „Attribute pro Textzeichen“.

H7: Struktur bringt unmittelbaren Nutzen für Benutzer.

H7.1: Verwendung der type tag Übersicht zur Navigation und zum Editieren (default views) Mittels einer Piwik-Auswertung kann gezeigt werden, dass zur type tag Übersichtstabelle navigiert wird.
Durch Auslesen der Handler kann darüber hinaus gezeigt werden, dass in der type tag Übersichtstabelle editiert wird. Das kann auch mit meinem Plugin geschehen.

H7.2: Attribute und Typen werden in eingebetteten Suchen verwendet. Es wird mit Hilfe der Attributfacette gesucht. Es wird mit Hilfe der type (tag) Facette gesucht. Diese Frage kann wiederum nur mittels Piwik-Auswertung erfolgen.

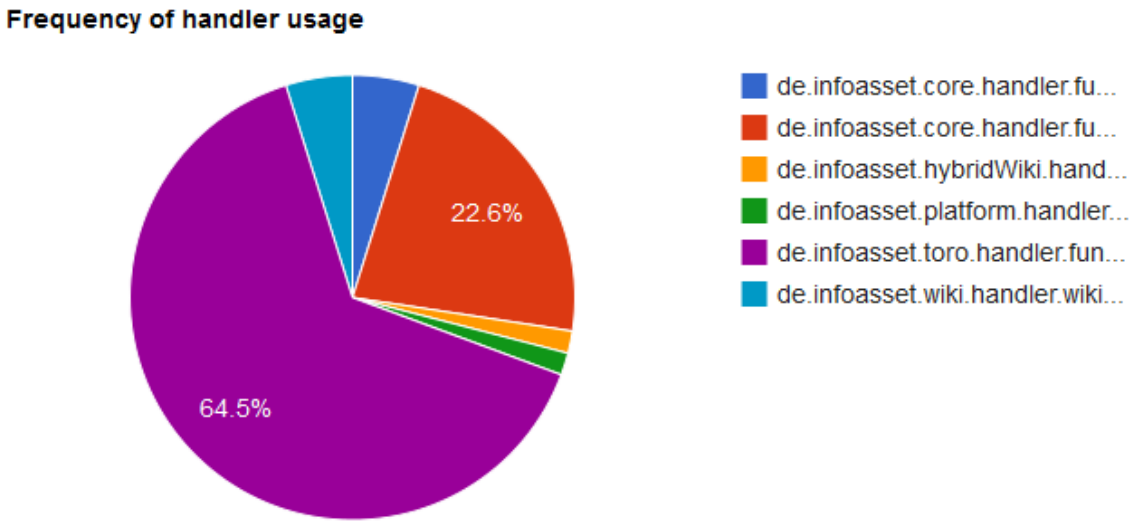


Abbildung 8.6: Diese Visualisierung zeigt, welche Handler an den Änderungen am strukturierten Teil der Wikiseite „Hybrid Wikis“ beteiligt waren. Mittels Mouseover wird der Handlername komplett angezeigt.

8.3 Auswertung und Interpretation der Ergebnisse

Im vorigen Abschnitt habe ich aufgezeigt, wie die Forschungsfragen und Hypothesen beantwortet werden können. Nun möchte ich an Hand des Dashboards die gewonnenen Ergebnisse detaillierter darstellen, so dass die eben skizzierten Lösungsansätze genauer ausgearbeitet werden.
Zunächst möchte ich die Analyse der gesamten Wikiseiten im WikiSpace „sebis Public Website“ vorstellen. Bei der Analyse von mehreren Wikiseiten gibt es (im Gegensatz zur Analyse einzelner Wikiseiten) ein spezielles Dashboard, das quantitative Abfrageergebnisse in Tabellenform darstellt (siehe Abb. 8.3) sowie grafische Visualisierungen. Zu Beginn werden im Dashboard die Anzahl der aktuellen Wikiseiten sowie die Anzahl der aktuellen Attribute und type tags genannt.

Im Fall der sebis Public Website waren zur Zeit des SQL-Datenbankdumps im November 2011 323 Wikiseiten mit zuletzt insgesamt 1471 Attributen. Die aktuelle Zahl der Attribute wurde durch die Tricia-Funktion

```
$xpath(xpath='count(/*/wikipage/snapshot[1]/attribute)')$
```

generiert. Es handelt sich dabei um eine einfache XPath-Abfrage, die zu einem beliebigen root Element (in diesem Fall usageAnalysis sämtliche Attribute innerhalb des ersten snapshot-Elements zurückgibt. Mittels der XPath-Funktion count() wird die Anzahl der matching nodes zurückgegeben. Wichtig zu wissen ist, dass die Reihenfolge der snapshots so definiert ist, dass das erste Element den aktuellen Zustand widerspiegelt, während das letzte Element den ersten snapshot darstellt. Die Sortierung der snapshots erfolgte folglich in umgekehrter chronologischer Reihenfolge.

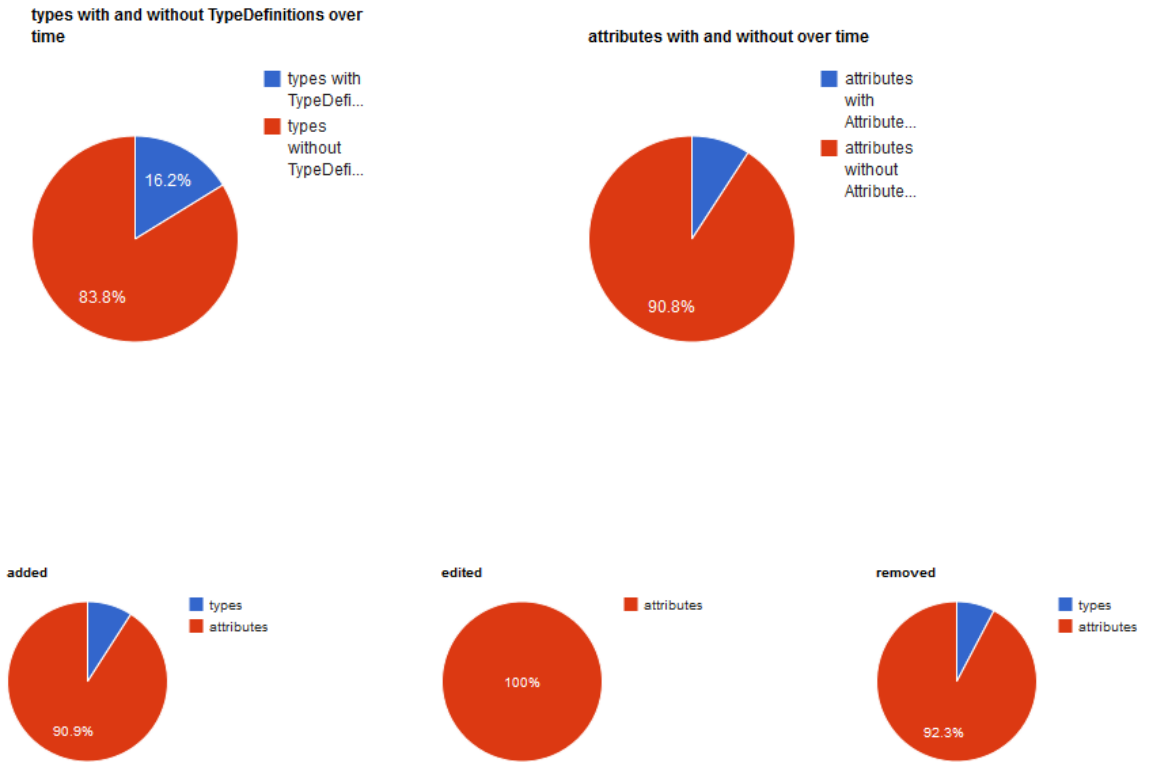


Abbildung 8.7: Diese Visualisierung der Wikiseite „Hybrid Wikis“ zeigt den Anteil der Typen und Attribute mit bzw. ohne Integritätsbedingungen sowie der Anteil von Typen bzw. Attributen bei Lösch- und Bearbeitungsvorgängen sowie Hinzufügeoperationen. Die Grafik, die den Anteil von Attributen und Typen an Bearbeitungsoperationen zeigt, hat per Definition zu 100% Attribute, da Typnamen nicht editiert werden können. Per Mouseover erhält man Auskunft über die jeweiligen Häufigkeiten.

Anschließend erscheint auf dem Dashboard eine vierspaltige Tabelle, die wie folgt aufgebaut ist:
(In Klammern steht dabei jeweils die Frage, die innerhalb der jeweiligen Spalte beantwortet werden soll.)

- in der ersten Spalte wird die XPath-Abfrage erklärt;
- in der zweiten Spalte wird das Ergebnis der XPath-Abfrage in Bezug auf den aktuellen snapshot dargestellt (Frage: Wie verhält es sich aktuell?)
- in der dritten Spalte wird das Ergebnis der XPath-Abfrage in Bezug auf alle snapshots in der Versionshistorie wiedergegeben (Frage: Hat es jemals einen solchen snapshot gegeben?)
- das Ergebnis der vierten Spalte bezieht sich auf den ersten erstellten snapshot, also den Zustand, als die Wikiseite erstellt wurde (Frage: War der snapshot bereits von

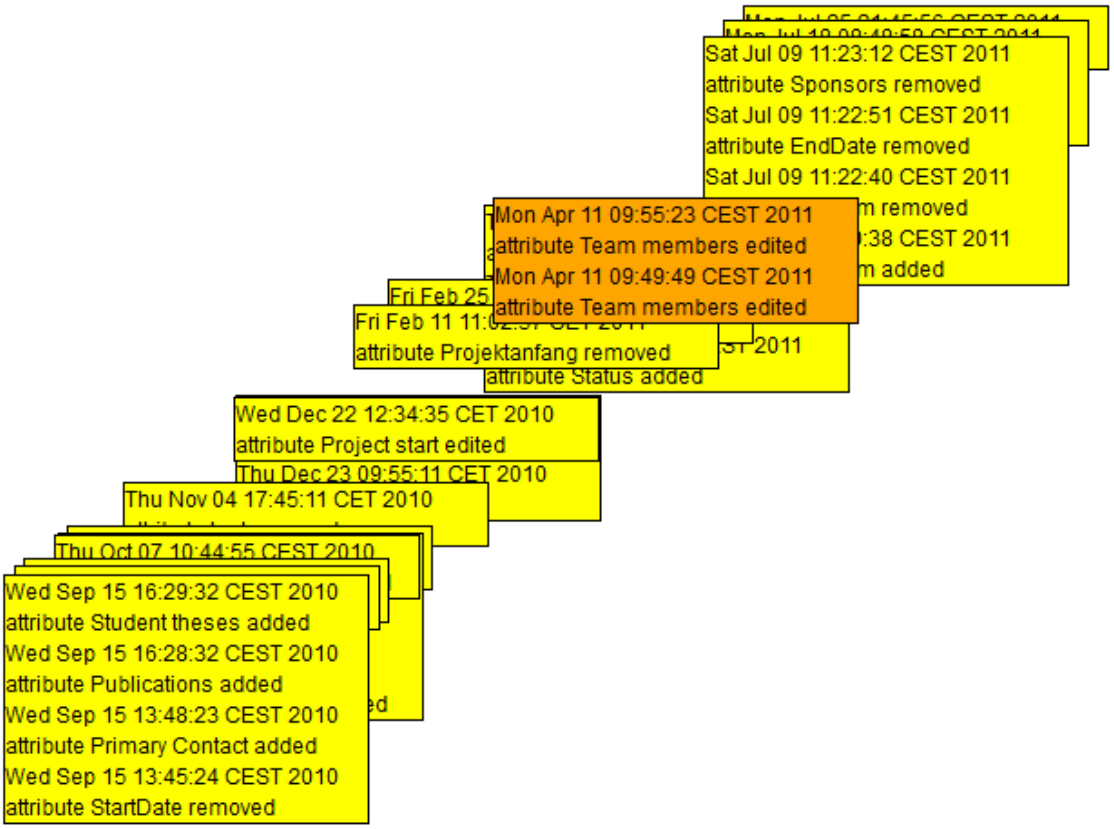


Abbildung 8.8: Diese von mir erstellte HTML-Visualisierung zeigt die Aktionen im Verlauf der Zeit, wobei die jeweiligen Boxen erst bei Mouseover angezeigt werden. Diese Visualisierung bezieht sich auf die Wikiseite „Hybrid Wikis“.

Anfang an so, wie in der XPath-Abfrage spezifiziert?)

Für die Funktionalität in Spalte 2 wird bei den jeweiligen Abfragen snapshot[1] verwendet, bei den Abfragen in Spalte 2 wird snapshot ohne Prädikate verwendet, während snapshot[last()] für die Auswahl des Zustands beim Erstellen der Wikiseite sorgt. Um die verwendeten XPath-Abfragen nachvollziehbarer werden zu lassen, habe ich per JavaScript eine MouseOver-Funktion entwickelt, die den verwendeten XPath-Befehl beim Überfahren einer Zahl einblendet.
Was in Abb. 8.3 auffällt ist, dass die Anzahl der Attribute von anfangs 387 auf aktuell 1471 stark angestiegen ist. Das ist ein Hinweis darauf, dass die Stuktur inkrementell und nicht ad hoc entsteht, also der Strukuturierungsprozess evolutionärer und weniger revolutionärer Natur ist.
Dies wird auch durch die in etwa Vervierfachung der Attributzahl pro Wikiseite verdeutlicht, die von 1,2 zu Beginn auf 4,6 am Ende ansteigt. Die Anzahl der Attribute ist allerdings nicht stetig steigend (was einer ständigen Zunahme der Struktur entsprechen würde), sondern Fluktuationen unterworfen. So ist der Durchschnittswert an Attributen in der Versionshistorie mit 5,6 höher als die aktuelle Zahl von Attributen. Dieser Wert be-

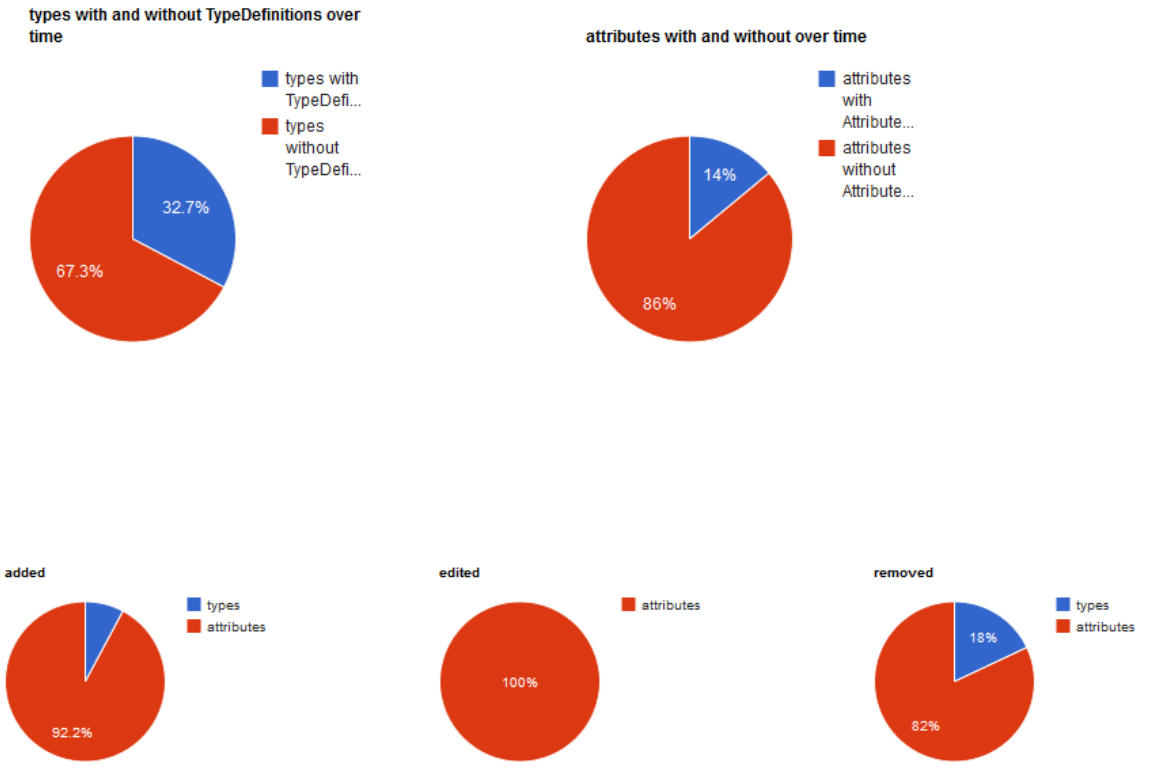


Abbildung 8.9: Diese Visualisierung des Wikis „sebis Public Website“ zeigt den Anteil der Typen und Attribute mit bzw. ohne Integritätsbedingungen sowie der Anteil von Typen bzw. Attributen bei Lösch- und Bearbeitungsvorgängen sowie Hinzufügeoperationen. Die Grafik, die den Anteil von Attributen und Typen an Bearbeitungsoperationen zeigt, hat per Definition zu 100% Attribute, da Typnamen nicht editiert werden können. Per Mouseover erhält man Auskunft über die jeweiligen Häufigkeiten.

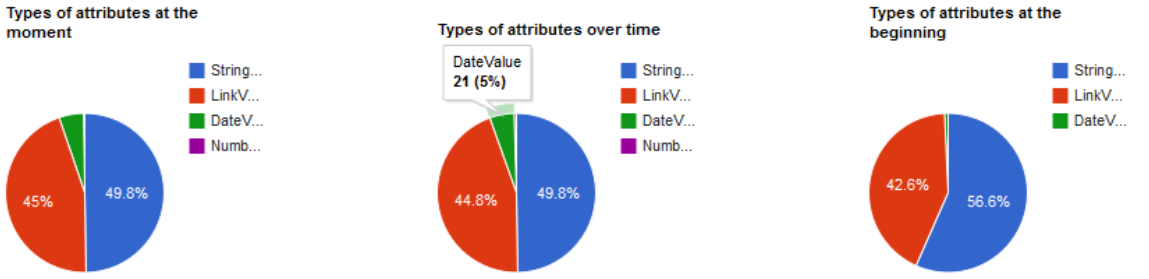


Abbildung 8.10: Verteilung von Attributtypen im Moment, in der Evolution des Wikis und zu Beginn in Bezug auf das Wiki „sebis Public Website“

rechnet sich aus der Anzahl von Attributen pro snapshot, wobei ein snapshot bei jeder Strukturänderung an Attributen und Typen angelegt wird.

number of type tags:	229 total at the moment
number of tags:	1445 total at the moment
number of WikiPages:	323
number of WikiPages where types were created before attributes	118
number of WikiPages where types were created after attributes	5
number of WikiPages where types were created at the same time as attributes	87

query	current	in version history	when page was created
number of attributes	1471	-	387
number of attributes per WikiPage	4.554179566563468	5.550656934306569	1.1981424148606812
number of attributes with AttributeDefinition per Wiki page	0.628482972136223	0.7757664233576642	0.20743034055727555
number of attributes per attribute with AttributeDefinition	7.246305418719212	7.155062100112909	5.776119402985074
number of WikiPages with attributes	209	210	81
number of WikiPages without attributes	114	113	242
number of attributes per type tag	4.554179566563468	5.550656934306569	1.1981424148606812
number of type tags	229	-	200
number of type tags per WikiPage	0.7089783281733746	0.9559124087591241	0.6191950464396285
number of WikiPages with type tags	209	211	188
number of WikiPages without type tags	114	112	135
number of WikiPages with attributes and without type tags	2	0	3
number of WikiPages without attributes and with type tags	2	1	110
number of WikiPages without attributes and without type tags	112	112	132
number of WikiPages with attributes and with type tags	207	210	78
number of WikiPages with attributes of type string	208	209	77
number of WikiPages with type tags and attributes of type string	207	209	74
number of WikiPages without attributes of type string	115	114	246
number of WikiPages with attributes of type link	188	188	58
number of WikiPages without attributes of type link	135	135	265
number of WikiPages with attributes of type currency	0	0	0
number of WikiPages with attributes of type dateValue	21	21	1
number of WikiPages with attributes of type number	1	2	0
number of WikiPages with attributes of type percentage	0	0	0
number of tags	1445	-	1331
number of tags per WikiPage	4.473684210526316	5.430656934306569	4.120743034055727
number of WikiPages with tags	274	274	270
number of WikiPages without tags	49	49	53

Abbildung 8.11: Quantitative Analyse der sebis Public Website

Da jedes Abspeichern einer Veränderung an den Strukturierungskonzepten einen neuen snapshot ergibt, kann man die hier gewonne Zahl auch als Anzahl von Attribute pro Modifikationsschritt an den Wikiseiten-eigenen Typen oder Attributen auffassen. Das schränkt allerdings die Aussagekraft dieser Zahl nicht ein. Wenn man bedenkt, dass die Anzahl der Attribute zu Beginn 1,2 betrug, aktuell 4,6 und im Versionsdurchschnitt 5,6 betrug, so ist die Abweichung des Versionsdurchschnitts vom aktuellen Durchschnitt durchaus signifikant. Immerhin liegt der Durchschnittswert circa 22% über dem aktuellen Wert. Das kann bedeuten, dass im Laufe der Versionsgeschichte weitaus mehr Attribute angelegt wurden, als sich später als sinnvoll erwiesen hat oder dass viele Wikiseiten neu erstellt wurden, wobei allerdings weniger Attribute pro Wikiseite angelegt wurden.
Die Anzahl der Wikiseiten mit Attributen beträgt aktuell 209, in der Versionshistorie 210, während es 81 Wikiseiten gab, die von Anfang an mit Attributen angelegt wurden. Das bedeutet, dass es durchaus nicht selten vorkommt, dass eine Wikiseite ohne Attribute später mit Attributen „nachgerüstet" wird.
Die Tatsache, dass 210 Seiten in früheren Versionen Attribute hatten, während aktuell „nur" 209 Wikiseiten Attribute haben kann nur daran liegen, dass die Attribute wieder gelöscht wurden. Es ist nicht möglich, dass die Wikiseite gelöscht wurde, da sie sonst nicht analysiert worden wäre. Welche Seite attributfrei geworden ist, könnte man mit einer speziellen XPath-Abfrage herausfinden.
Interessant ist auch die Anzahl der Wikiseiten ohne Attribute. Die Ergebnisse an sich sind dabei weniger überraschend, weil sich die Ergebnisse auch aus der Differenz zwischen der Gesamtzahl der Wikiseiten und den vorigen Ergebnissen berechnen lassen. Vielmehr ist die Sicht aus der anderen Perspektive interessant. Die besagt nämlich, dass 113 Wikiseiten von 323, also gut ein Drittel, nie mit Attributen „in Kontakt" gekommen sind. Das empfinde ich als relativ viel, zumal es sich bei einem Hybrid Wiki um eine Software handelt, deren Stärke vor allem in der Speicherung strukturierter Informationen liegt.
Nun stellt sich natürlich die Frage, wie es mit Wikiseiten und deren Typen aussieht. Es stellt sich heraus, dass sich die Zahlen beinahe identisch verhalten wie die Quantitäten der Attribute. Es gibt 209 Wikiseiten (gleiche Zahl bei Attributen), die aktuell Typen besitzen, folglich besitzen 114 Wikiseiten keine Typen (bzw. Attribute). In der Versionshistorie sieht es ähnlich aus: 211 Wikiseiten mit Typen, während es 210 Wikiseiten mit Attributen gibt. Der Verdacht, dass aufgrund der Nähe der beiden Zahlen zueinander ein Zusammenhang besteht, bestätigt sich: Es gibt 112 Seiten die aktuell und 112 Seiten die in der Versionshistorie weder Attribute noch Typen hatten. In anderen Worten bedeutet das, dass circa ein Drittel der Seiten keinerlei Strukturierungskonzepte verwenden. Es ist relativ selten, dass nur eine Art von Strukturierungskonzept verwendet wird, während die andere Art von Strukturierungskonzept nicht verwendet wird.
An Wikiseiten die aktuell Attribute verwenden, aber keine Typen, gibt es aktuell zwei. Der umgekehrte Fall in dem Typen aber keine Attribute verwendet werden tritt bei zwei Wikiseiten auf. Es gibt eine Wikiseite, die in ihrer gesamten Versionshistorie keine Attribute sondern nur Typen verwendet hat, während der umgekehrte Fall gar nicht auftritt: Werden also Attribute verwendet, so gab es mindestens einen Zeitpunkt, an dem Typen existiert haben.
Erstaunlicherweise hat es sich bei vielen Seiten bereits eingebürgert, zu Beginn Typen aber keine Attribute anzulegen. Dieser Fall trat mit 110 Wikiseiten bereits bei grob einem Drittel aller Wikiseiten auf. Der umgekehrte Fall, dass Attribute von Anfang an angelegt wurden,

gleichzeitig aber keine Typen gewählt wurden, trat nur dreimal auf.
Mit Attributen und Typen wurden 78 Wikiseiten bereits zu Beginn angelegt. Bei 210 Seiten hat es zu einem Zeitpunkt sowohl Attribute als auch Typen gegeben, während das aktuell noch bei 207 Wikiseiten der Fall ist. Man erkennt also, dass der Konvergenzzustand einer Wikiseite nicht vollständige Strukturierung allein mittels Typen und Attributen ist, da es immerhin Gegenbeispiele gibt, bei denen man sich entschlossen hat, auf Typen bzw. Attribute zu verzichten.

Wie man sieht, ermöglichen bereits die Standardvisualisierungen zahlreiche Erkenntnisse bezüglich der Nutzung von hybriden Wikis und der Evolution von Strukturierungskonzepten. Mit Hilfe des Plugins können auch andere Tricia-Installationen überprüft werden, um Ähnlichkeiten und Unterschiede im Nutzungsverhalten zu identifizieren.

9 Fazit

Das entwickelte Tricia-Plugin stellt ein vielseitiges Werkzeug zur Visualisierung der Evolution der Strukturierungskonzepte in hybriden Wikis dar, das neben vordefinierten Abfragen, die in Form eines Dashboards zur Verfügung gestellt werden, auch eigene Visualisierungen und Abfragen ermöglicht. Es gibt Dashboards für einzelne Wikiseiten, mehrere Wikiseiten (etwa alle Wikiseiten aus einem Wikispace) sowie die Möglichkeit, Typ- und Attributdefinitionen eines Wikispace analysieren zu lassen. Die jeweiligen Standard-Visualisierungen sind dabei stets speziell auf die zu analysierenden Entities zugeschnitten; das heißt für eine einzelne Wikiseite gibt es eine andere Visualisierung als für mehrere Wikiseiten, etc.
Darüber hinaus besteht die Möglichkeit, eigene Visualisierungen mittels Tricia-Script zu erstellen, die mit Hilfe eines Wrappers Google-Chart-Visualisierungen ermöglichen. Abfrageergebnisse können dabei auch in Form einer ListSubstitution oder PrintSubstitution in die eigene Seite eingebunden werden.

Limitierungen Das System ist darin limitiert, dass lediglich Wikiseiten analysiert werden, die zum Zeitpunkt der Analyse existierten; gelöschte Wikiseiten werden nicht berücksichtigt, auch wenn das für eine Wikispace-Analyse interessant sein kann. Darüber hinaus werden von meinem Plugin Attributdefinitionen, die nie verwendet wurden und im aktuellen Zustand nicht mehr existieren (also gelöscht wurden), nicht erfasst. Das Plugin um diese Fähigkeit zu erweitern wäre allerdings wenig arbeitsaufwendig.
Aus sicherheitstechnischen Aspekten wurde die Anzahl der jeweils gleichzeitig ablaufenden Abfragen auf eine beschränkt, um eine Überlastung des Systems durch Angreifer zu vermeiden und da mit dieser Entscheidung eine effektive Nutzung eines Caches erst möglich wurde. Dieser Cache dient dazu, die Abfragegeschwindigkeit soweit zu erhöhen, dass die Software produktiv eingesetzt werden kann.

Auswertung der Analyse Auch wenn die zur Verfügung stehenden Daten nur eingeschränkt aussagekräftig sind, so ist doch erkennbar, dass die Strukturierung inkrementell erfolgt. Es ist zu berücksichtigen, dass der Lehrstuhl, der hybride Wikis entwickelt hat, diese möglicherweise vorbildlicher verwendet als Laien. Basierend auf dem entstandenen Tool kann nun eine weitere Arbeit klären, wie die Strukturevolution bei anderen Unternehmen, die hybride Wikis einsetzen, verlaufen ist; dies ist durch Integration meines Plugins in deren Tricia-System ohne Installation von zusätzlicher Software möglich.

Appendix

10 XML Schema des generierten XML-Dokuments

Listing 10.1: XML-Schema des generierten XML-Dokuments

```
<?xml version="1.0" encoding="utf8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="idtype">
    <xs:restriction base="xs:string">
      <xs:pattern value="wikiPage/[a-z0-9]+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="truefalse">
    <xs:restriction base="xs:string">
      <xs:enumeration value="true"/>
      <xs:enumeration value="false"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="attributeDefinition">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="type" type="xs:string" minOccurs="0"
            maxOccurs="unbounded" />
        <xs:element name="typeClass" type="xs:string" minOccurs
            ="0" maxOccurs="1" />
        <xs:element name="linkType" type="xs:string" minOccurs
            ="0" maxOccurs="unbounded" />
        <xs:element name="defaultValue" type="xs:string"
            minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="multiplicity" type="xs:string"
            minOccurs="0" maxOccurs="1" />
        <xs:element name="strict" type="truefalse" minOccurs="1"
            maxOccurs="1" />
        <xs:element name="description" type="xs:integer"
            minOccurs="1" maxOccurs="1" />
      </xs:sequence>
      <xs:attribute name="when_text" type="xs:string" />
      <xs:attribute name="when_long" type="xs:integer" />
      <xs:attribute name="days" type="xs:integer" />
      <xs:attribute name="key" type="xs:string" />
```

```
29        <xs:attribute name="id" type="xs:string" />
30      </xs:complexType>
31    </xs:element>
32    <xs:element name="attribute">
33      <xs:complexType>
34        <xs:sequence>
35          <xs:element name="key" type="xs:string" minOccurs="1"
              maxOccurs="1" />
36          <xs:element name="value" minOccurs="0" maxOccurs="
              unbounded" >
37            <xs:complexType>
38              <xs:simpleContent>
39                <xs:extension base="xs:string">
40                  <xs:attribute name="type" type="xs:string" use="
                      optional" />
41                </xs:extension>
42              </xs:simpleContent>
43            </xs:complexType>
44          </xs:element>
45          <xs:element ref="attributeDefinition" minOccurs="0" />
46        </xs:sequence>
47      </xs:complexType>
48    </xs:element>
49    <xs:element name="typeDefinition">
50      <xs:complexType>
51        <xs:sequence>
52          <xs:element name="strict" type="truefalse" minOccurs="1"
              maxOccurs="1" />
53          <xs:element name="description" type="xs:integer"
              minOccurs="1" maxOccurs="1" />
54        </xs:sequence>
55        <xs:attribute name="when_text" type="xs:string" />
56        <xs:attribute name="when_long" type="xs:integer" />
57        <xs:attribute name="days" type="xs:integer" />
58        <xs:attribute name="key" type="xs:string" />
59        <xs:attribute name="id" type="xs:string" />
60      </xs:complexType>
61    </xs:element>
62    <xs:element name="type">
63      <xs:complexType mixed="true">
64        <xs:sequence>
65          <xs:element ref="typeDefinition" minOccurs="0" maxOccurs
              ="1" />
66        </xs:sequence>
67      </xs:complexType>
68    </xs:element>
```

```
69  <xs:element name="params">
70    <xs:complexType mixed="true">
71      <xs:sequence>
72        <xs:element name="param" minOccurs="0" maxOccurs="
              unbounded" >
73          <xs:complexType>
74            <xs:simpleContent>
75              <xs:extension base="xs:string">
76                <xs:attribute name="id" type="idtype" use="
                      required" />
77              </xs:extension>
78            </xs:simpleContent>
79          </xs:complexType>
80        </xs:element>
81      </xs:sequence>
82    </xs:complexType>
83  </xs:element>
84  <xs:element name="function">
85    <xs:complexType mixed="true">
86      <xs:sequence>
87        <xs:element name="name" type="xs:string" minOccurs="0"
              maxOccurs="1" />
88        <xs:element ref="params" minOccurs="0" maxOccurs="
              unbounded" />
89        <xs:element name="block" type="xs:string" minOccurs="0"
              maxOccurs="unbounded" />
90      </xs:sequence>
91    </xs:complexType>
92  </xs:element>
93  <xs:complexType name="changedtype">
94    <xs:sequence>
95      <xs:element ref="attribute" minOccurs="0" maxOccurs="
            unbounded" />
96      <xs:element ref="type" minOccurs="0" maxOccurs="unbounded"
            />
97    </xs:sequence>
98  </xs:complexType>
99  <xs:element name="diff">
100   <xs:complexType>
101     <xs:sequence>
102       <xs:element name="added" type="changedtype" minOccurs="0"
              maxOccurs="unbounded" />
103       <xs:element name="edited" type="changedtype" minOccurs
              ="0" maxOccurs="unbounded" />
104       <xs:element name="removed" type="changedtype" minOccurs
              ="0" maxOccurs="unbounded" />
```

```
105         <xs:element name="action" type="xs:string" minOccurs="0"
                maxOccurs="unbounded" />
106       </xs:sequence>
107       <xs:attribute name="old_when_text" type="xs:string" />
108       <xs:attribute name="old_when_long" type="xs:integer" />
109       <xs:attribute name="old_days" type="xs:integer" />
110       <xs:attribute name="new_when_text" type="xs:string" />
111       <xs:attribute name="new_when_long" type="xs:integer" />
112       <xs:attribute name="new_days" type="xs:integer" />
113     </xs:complexType>
114   </xs:element>
115   <xs:element name="snapshot">
116     <xs:complexType>
117       <xs:sequence>
118         <xs:element name="content" type="xs:integer" minOccurs
                ="0" maxOccurs="1" />
119         <xs:element name="handler" type="xs:string" minOccurs="0"
                 maxOccurs="1" />
120         <xs:element ref="attribute" minOccurs="0" maxOccurs="
                unbounded" />
121         <xs:element ref="type" minOccurs="0" maxOccurs="unbounded
                " />
122         <xs:element name="tag" type="xs:string" minOccurs="0"
                maxOccurs="unbounded" />
123         <xs:element ref="diff" minOccurs="0" maxOccurs="unbounded
                " />
124       </xs:sequence>
125       <xs:attribute name="when_text" type="xs:string" />
126       <xs:attribute name="when_long" type="xs:integer" />
127       <xs:attribute name="days" type="xs:integer" />
128     </xs:complexType>
129   </xs:element>
130
131   <xs:element name="diff2">
132     <xs:complexType>
133       <xs:sequence>
134         <xs:any minOccurs="0" maxOccurs="unbounded"
                processContents="skip" />
135       </xs:sequence>
136       <xs:attribute name="when_text" type="xs:string" />
137       <xs:attribute name="when_long" type="xs:integer" />
138       <xs:attribute name="days" type="xs:integer" />
139     </xs:complexType>
140   </xs:element>
141
142   <xs:element name="wikipage">
```

```
143     <xs:complexType>
144       <xs:sequence minOccurs="0" maxOccurs="unbounded">
145         <xs:element name="hideNameOnPage" type="xs:string"
              minOccurs="0" maxOccurs="1" />
146         <xs:element name="parent" type="xs:string" minOccurs="0"
              maxOccurs="1" />
147         <xs:element name="child" type="xs:string" minOccurs="0"
              />
148         <xs:element name="scriptable" type="truefalse" minOccurs
              ="0" maxOccurs="1" />
149         <xs:element ref="function" minOccurs="0" maxOccurs="
              unbounded" />
150         <xs:element ref="snapshot" minOccurs="0" maxOccurs="
              unbounded" />
151         <xs:element ref="diff2" minOccurs="0" maxOccurs="
              unbounded" />
152       </xs:sequence>
153       <xs:attribute name="id" type="idtype" use="required" />
154       <xs:attribute name="url" type="xs:string" />
155       <xs:attribute name="when_text" type="xs:string" />
156       <xs:attribute name="when_long" type="xs:integer" />
157       <xs:attribute name="days" type="xs:integer" />
158     </xs:complexType>
159   </xs:element>
160   <xs:element name="usageAnalysis">
161     <xs:complexType>
162       <xs:sequence>
163         <xs:element ref="wikipage" maxOccurs="unbounded" />
164       </xs:sequence>
165     </xs:complexType>
166   </xs:element>
167 </xs:schema>
```

11 Grundlegende XPath-Sprachkonstrukte

Mein Tricia-Plugin stellt Funktionen zur Verfügung, um per XPath Abfragen auf dem generierten XML-Dokument auszuführen. Aus diesem Grund werden im Folgenden kurz und knapp die wichtigsten XPath-Funktionen basierend auf [BBC+12] vorgestellt. Die Beispiele werden sich dabei auf folgende Beispiel-XML-Datei beziehen:

```
<?xml version="1.0" encoding="utf8" ?>
<root>
    <child>text1</child>
    <child>text2</child>
    <child>
        <other attr="attr1">text3a</other>
        <other attr="attr2">text3b</other>
    </child>
    <child>text4</child>
</root>
```

XML stellt eine Baumstruktur mit einem einzigen Root-Element dar. Diese hierarchisch strukturierten XML-Elemente können per Pfad adressiert werden. Absolute Pfade, beginnen in XPath mit / (Schrägstrich), gefolgt vom Namen des root-Elements.

Beispiel 11.1 `/root/child/other` *liefert text3a, text3b*

Wenn das gewünschte XML-Element kein direktes Kind, sondern auch ein „Enkel", „Urenkel"etc. sein kann, so kann mit // (zwei Schrägstriche) ein recursive descent selektiert werden.

Beispiel 11.2 `/root//other` *liefert text3a, text3b da sie* recursive descendants *des* root*-Elements sind.*

Relative Pfade beziehen sich jeweils auf die aktuelle context node und beginnen mit . (Punkt).

Beispiel 11.3 `./child/other` *liefert text3a, text3b nur, wenn die* context node `/root` *gewählt wurde, sonst wird* `false` *zurückgegeben.*

Um das parent-Element zu adressieren kann man .. (zwei Punkte) verwenden.

Beispiel 11.4 *Wenn als* context node `/root/child/other[1]` *gewählt wurde, liefert* `../child` *alle vier* XML*-Elemente mit dem* NodeName *child.*

Wie man sieht, lehnt sich die Pfadangabe an die in Windows- und Unix-Dateisystemen üblichen Pfade an. Vor allem `.` (aktuelles Verzeichnis) und `..` (zur übergeordnetes Verzeichnis) sind in Dateisystemen geläufig.

Attribute werden mittels @ adressiert:

Beispiel 11.5 `/root/child/other/@attr` *liefert attr1 und attr2*

Wildcards wie `*` (Stern) oder `*@` referenzieren jedes beliebige XML-Element bzw. Attribut.

Beispiel 11.6 `/*/child` *liefert die* child node*, egal welchen* NodeName *das* root*-Element trägt.*

Prädikate stellen Forderungen an die gewählten Elemente und fungieren somit als Filter. Prädikate werden in eckige Klammern geschrieben. Innerhalb der eckigen Klammern können mehrere Arten von Prädikaten verwendet werden. Wird in eckigen Klammern eine Zahl n gestellt, so wird das n-te XML-Element selektiert. Auch wenn in vielen Programmiersprachen mit [0] das erste array-Element gemeint ist, existiert in der XPath-Spezifikation das „nullte“ Element nicht. Stattdessen wird das erste Element mit [1] selektiert.

Beispiel 11.7 /root/child[3]/other[2] *liefert text3b, also das dritte* child XML*-Element und dann das zweite* other XML*-Element.*

Prädikate Im Prädikat können auch Funktionen und Filterausdrücke verwendet werden.

Beispiel 11.8 *Prädikate*

- `/root/child[last()]` *gibt das letzte* child*-Element unterhalb des* root*-Elements zurück.*
- `/root/child[position() < 3]` *gibt die ersten zwei* child*-Elemente zurück.*
- `/root/child[not(./other)]` *gibt alle* child*-Elemente zurück, die keine* other*-Elemente enthalten.*
- `//other[@attr="attr2"]` *gibt die/das* other-XML*-Element(e) zurück, deren Attributwert attr den Wert attr2 aufweist. In diesem Fall wird also nur das zweite* other-XML*-Element zurückgegeben.*

Prädikate können auch logisch verknüpft werden.

Beispiel 11.9 *Logisch verknüpfte Prädikate*

- `/root/child[(position() < 3) or (./other)]` *gibt die ersten drei* child*-Elemente zurück, wobei die ersten beiden mittels* `position() < 3` *ausgewählt werden und das dritte durch die Abfrage* `./other` *selektiert wird. Dabei bedeutet* `./other`*, dass jedes* child-XML*-Element, das ein* other-XML*-Element enthält, selektiert wird.*

XPath-Funktionen Interessant sind vor allem die Funktionen `count(*)`, `not(*)` sowie `text()`, `last()` und `position()` .

Mit der `count(*)`-Funktion wird die Anzahl der selektierten XML-Dokumente berechnet (in diesem Fall alle XML-Elemente in root-Ebene, da der Selektor `*` verwendet wird). `not(*)` negiert den Ausdruck, würde also in diesem Fall `false` zurückgeben, da `*` (Stern) das root-Element wählt und die Negierung `false` liefert.

Beispiel 11.10 *XPath-Funktionen*

- `/root/child/text()` *gibt ganz speziell die gewählte* text node *aus, wählt also nicht das eigentliche* XML*-Element, sondern dessen Textinhalt.*
- `count(//@attr)` *gibt die Anzahl aller* XML*-Elemente im gesamten* XML*-Dokument an.*

In eckigen Klammern (Prädikate) stehen Forderungen an die gewählten Elemente und fungieren somit als Filter.

XPath-Operatoren Als logische Operatoren stehen beispielsweise `and` und `or` als Vergleichsoperatoren <, >, <=, >= zur Verfügung.

Error handling Solange eine XPath-Abfrage syntaktisch korrekt ist, wird ein gültiges Ergebnis ausgegeben. Wenn der Pfad nicht lokalisiert werden kann, wird `false` (und kein Fehler) zurückgegeben. Das ist hilfreich, da auf diese Weise XPath-Abfragen sehr robust sind und bei einer Änderung der zugrundeliegenden XML-Datei lediglich `false` zurückgegeben wird, anstatt mit Fehlern den Programmablauf zu unterbrechen.

12 Abkürzungen

DMKD Data Mining und Knowledge Discovery

XML Extensible Markup Language

W3C World Wide Web Consortium

JSON JavaScript Object Notation

ORM Object Relational Mapping

Literaturverzeichnis

[BBC+12] BERGLUND, Anders ; BOAG, Scott ; CHAMBERLIN, Don ; FERNÁNDEZ, Mary F. ; KAY, Michael ; ROBIE, Jonathan ; SIMÉON, Jérôme: *XML Path Language (XPath) 2.0 (Second Edition)*. `http://www.w3.org/TR/xpath20/`. Version: April 2012

[BPSM+08] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, C. M. ; MALER, Eve ; YERGEAU, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. `http://www.w3.org/TR/2008/REC-xml-20081126/`. Version: November 2008

[CK05] CIOS, Krzysztof ; KURGAN, Lukasz: Trends in Data Mining and Knowledge Discovery. In: *In: Pal N.R., Jain, L.C. and Teoderesku, N. (Eds.), Knowledge Discovery in Advanced Information Systems*, Springer, 2005, S. 200–2

[CVK98] CHERVENAK, Ann ; VELLANKI, Vivekanand ; KURMAS, Zachary: Protecting File Systems: A Survey of Backup Techniques. In: *Joint NASA and IEEE Mass Storage Conference*, 1998

[Edm03] EDMONDS, Andrew N.: On data mining tree structured data in XML. In: *On data mining tree structured data in XML', Data mining UK conference, University of Nottingham, Aug 2003*, 2003

[GKPW03] GOTTLOB, Georg ; KOCH, Christoph ; PICHLER, Reinhard ; WIEN, Technische U.: *The Complexity of XPath Query Evaluation*. 2003

[Goo12] GOOGLE, Inc.: *Display live data on your site*. `https://developers.google.com/chart/`. Version: April 2012

[LR07] LANGENSCHEIDT-REDAKTION: *Langenscheidt Taschenwörterbuch Englisch-Deutsch:*. Langenscheidt, 2007. – ISBN 9783468101328

[Mat11a] MATTHES, Florian: *Basics (types)*. `http://www.infoasset.de/wikis/tricia-help/basics-types`. Version: Oktober 2011

[Mat11b] MATTHES, Florian: *Class Plugin*. `http://www.infoasset.de/wikis/javadoc-import-wiki/platform-services-plugin`. Version: November 2011

[Mat12a] MATTHES, Florian: *Class Handler in package de.infoasset.platform.handler*. `http://www.infoasset.de/wikis/javadoc-import-wiki/platform-handler-handler`. Version: April 2012

[Mat12b] MATTHES, Florian: *Class ObjectRelationalMappingDoc in package de.infoasset.platform.documentation*. `http://www.infoasset.de/wikis/javadoc-import-wiki/platform-documentation-objectrelationalm`. Version: April 2012

[MNS11] MATTHES, Florian ; NEUBERT, Christian ; STEINHOFF, Alexander: Hybrid Wikis: Empowering Users to Collaboratively Structure Information. In: *6th International Conference on Software and Data Technologies (ICSOFT), Seville*, 2011

[Neu12] NEUBERT, Christian: *Hybrid Wikis - Enterprise Collaboration and Information Management*. `http://wwwmatthes.in.tum.de/wikis/sebis/hybrid-wiki`. Version: April 2012

[Ora12] ORACLE: *XML Functions*. `http://dev.mysql.com/doc/refman/5.5/en/xml-functions.html`. Version: April 2012

[Sla08] SLAVĚTÍNSKÝ, Václav: *Vizualizace XML schémat*, Bachelorarbeit, VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE, Fakulta informatiky a statistiky, Katedra informačního a znalostního inženýrství, November 2008. `http://sourceforge.net/projects/xsdvi/`

[Ull07] ULLENBOOM, Christian: *Java ist auch eine Insel*. 6., aktualisierte und erweiterte Auflage. Bonn : Galileo Computing, 2007 `http://www.galileocomputing.de/openbook/javainsel6/`

[Wie11] WIESI, Stefan: *Identifikation von Bedienbarkeitsproblemen in einem Enterprise 2.0 Werkzeug*, Bachelorarbeit, Fakultät für Informatik der Technischen Universität München, September 2011