Endbericht

PG 540 Semantic oDOBS

19. November 2010

Teilnehmer

Martin Adomat
Christian Altrogge
Mitja Bamberger
Thorsten Flick
Jens Hornung
Jakob Langer
Benedikt Mättig
Christoph Schlüter
Christian Winter
Yu Xue

Betreuer

Prof. Dr. Bernhard Steffen
Dipl. Inform. Ralf Nagel
Dipl. Inform. Sven Jörges

Inhaltsverzeichnis

Teil I.

Einführung

1. Einleitung

Der vorliegende Endbericht bietet einen Überblick über die Literaturdatenbank oCIOBS und den dort getätigten Änderungen. Das einleitende Kapitel befasst sich zunächst mit der Geschichte dieser Literaturdatenbank und motiviert das Thema daraufhin, indem oCIOBS mit den Alternativsystemen verglichen wird, woraus sich die Ziele der *Projektgruppe 540: Semantic* oCIOBS ergeben.

1.1. Grundbegriffe

Der Endbericht wird an vielen Stellen auf eine Terminologie zurückgreifen, die im Bereich der Bibliografien verwendet wird. Für Leser, die mit der Thematik nicht vertraut sind, erläutert dieser Abschnitt daher die wichtigsten Ausdrücke.

Der Begriff *Bibliografie* bezeichnet ein Verzeichnis von Literaturangaben. Diese Literaturangaben sind insbesondere für die wissenschaftliche Arbeit wichtig, denn sie erlauben die Recherche und das einfache Erstellen eines Literaturverzeichnisses bei der Anfertigung neuer Veröffentlichungen. Dementsprechend bietet ein *Bibliografiedienst* oder *-service* den Zugriff auf diese Informationen an und ermöglicht es, den Datenbestand zu durchsuchen.

Die Literaturangaben, die in den Bibliografien enthalten sind, können drei verschiedenen Publikationsarten zugeordnet werden. Zunächst seien die *Monografien* genannt. Dabei handelt es sich um eigenständige Werke eines (meist) einzigen Autors. Typischerweise ist dies ein Buch. Besonders für den Bereich der Informatik haben aber die beiden weiteren Typen – *Fachzeitschriften* und *Konferenz-/Tagungsbände* – eine große Bedeutung. Bei den Fachzeitschriften (engl. *journals*) handelt es sich um regelmäßig erscheinende Zeitschriften, die sich einem Spezialthema eines wissenschaftlichen Fachbereichs widmen. Konferenzbände (engl. *proceedings*) enthalten die schriftlichen Ausarbeitungen (engl. *inproceedings* genannt) der Vorträge einer Konferenz.

Bei den Verfassern von Publikationen werden *Autoren*, *Koautoren* und *Editoren* unterschieden. Die Autoren sind die Urheber einer Veröffentlichung. Sind mehrere von ihnen an einem Werk beteiligt, so sind aus der Sicht eines Autors die anderen Autoren seine Koautoren. Der Begriff Editor tritt in Zusammenhang mit Konferenzbänden auf und wird nicht immer einheitlich verwendet. Meist sind die Organisatoren der zugrunde liegenden Konferenz gemeint, es können aber auch die Personen angegeben sein, die aus verschiedenen Einzelbeiträgen einen Konferenzband erstellt

haben [11] [63].

1.2. Geschichte

oCOBS ist die Kurzform für den Dortmunder Online-Bibliografieservice, der am Lehrstuhl für Programmiersysteme[1] der Universität Dortmund im Rahmen der Projektgruppen 494 und 513 entwickelt wurde. Die PG 540 ist somit die Dritte, die sich die Weiterentwicklung und Verbesserung von oCOBS als Ziel gesetzt hat. Der Grundstein für das Projekt wurde 2006/2007 von der *PG 494 L2EE: Lightweight Process Coordination & J2EE* gelegt. Hinter dem Projekt steckt der Gedanke, eine anwenderfreundliche Datenbank für die Literatursuche in Publikationen zum Thema Informatik bereitzustellen.

Die Literaturbasis wird initial aus einer XML-Datei des „Digital Bibliography & Library Project“, kurz *DBLP* [vgl. Kapitel 7.1.1 auf Seite 40], importiert und in einer relationalen Datenbank gespeichert. Zusätzlich können auch manuell neue Einträge hinzugefügt oder bestehende Daten gepflegt werden. Es besteht außerdem die Möglichkeit weitere Daten von dritten Anbietern, z.B. im BibTeX-Format, zu importieren.

Die zweite *PG 513 DoPAC: Dortmund Online Public Access Catalog* startete 2007 und betrachtete unter anderem Performanzaspekte der Webanwendung, wodurch beispielsweise die Suchfunktion deutlich beschleunigt werden konnte. Zusätzlich wurde der Funktionsumfang erweitert. Beispielsweise wurde eine automatische Doublettensuche, die doppelt erfasste Publikationen findet und meldet, eingebaut. Für tiefer gehende Informationen sei an dieser Stelle auf die Endberichte der PG 494 [8] und 513 [4] verwiesen.

Auf dieser Grundlage basiert die Arbeit der *Projektgruppe 540: Semantic* oCOBS, deren Motivation im Folgenden erörtert wird.

1.3. Motivation

Für die Anfertigung von wissenschaftlichen Arbeiten ist es unerlässlich fachspezifische Literatur zu recherchieren. Sei es ein Student, der eine Semiar- oder Diplomarbeit verfassen muss, ein Wissenschaftler, der seine Arbeit untermauern und veröffentlichen will oder einfach alle Personen, die ihr Wissen mittels „korrekten“ und vollständigen Informationen erweitern möchten. Allen genannten Personen stellen sich die gleiche Frage: „Wo finde ich möglichst schnell und vollständig zitierfähige Literatur?“ Die gängigen und bekannten Anlaufstellen sind Bibliotheken, Internetsuchmaschinen oder Personen mit Expertenwissen. Alle drei Genannten haben

[1] `http://ls5-www.cs.tu-dortmund.de`

jedoch Nachteile, Bibliotheken haben beispielsweise das Problem, dass sie ortsgebunden sind, evtl. keinen Anspruch auf Vollständigkeit erheben oder Bücher schlichtweg entliehen sind. Suchmaschinen im Internet sind dagegen zwar nicht ortsgebunden und bieten eine Vielzahl von möglichen Treffern, haben aber den Nachteil, dass diese oft unsortiert und selten wissenschaftlich sind. Beim Expertenwissen stellen sich die Fragen woher die Person das Wissen hat, ob es inhaltlich „korrekt“ ist und vor allem ob es zitiert werden kann und darf. Das Fazit aller drei Ansätze ist: Literaturrecherche ist schwierig und zeitaufwendig.
Zur Lösung dieses Problems gibt es Suchdienste, die speziell auf wissenschaftliches Arbeiten ausgerichtet sind. Im Folgenden wird zunächst das DBLP der Universität Trier vorgestellt, welches auf den Bereich der Informatikliteratur ausgerichtet ist und die Grundlage für oCOBS bildet (vgl. Kapitel 1.2 auf der vorherigen Seite). Im Anschluss daran werden die etablierten Suchdienste CiteSeer^x, CiteULike und Google Scholar vorgestellt und verglichen.

1.3.1. DBLP

Das *Digital Bibliography & Library Project (DBLP)* [21] ist eine Sammlung von bibliografischen Metainformationen über Veröffentlichungen aus dem Bereich der Informatik, die an der Universität Trier von Dr. Michael Ley betrieben wird. Aus dessen Doktorarbeit ist der Dienst bereits 1993 hervorgegangen. Damals beschränkte sich der Inhalt auf den Bereich „Datenbanksysteme“ und „Logik Programmierung“, worauf auch der ursprüngliche Name *DataBase systems and Logic Programming (DBLP)* zurückgeht. Bis heute wurde der Inhalt aber auf das gesamte Feld der Informatik ausgedehnt. Aktuell[2] sind mehr als 1,4 Millionen Publikationen in der Datenbank gelistet. Die Besonderheit des DBLPs ist, dass alle Metadaten der Publikationen von Hand erfasst werden, was für eine herausragende Datenqualität sorgt. Dazu zählt beispielsweise auch die richtige Zuordnung von Publikationen zu den Konferenzbänden in denen sie veröffentlicht wurden. Auch die korrekte Behandlung von Personennamen, was gerade bei Autoren aus dem asiatischen Raum oder abgekürzten Zweitnamen oft zu Problemen führt, ist den Betreibern des DBLPs besonders wichtig, denn es werden verschiedene Personen mit dem gleichen Namen getrennt behandelt.

Das DBLP hat aber auch einige Nachteile. Beispielsweise bietet es für die Benutzer lediglich grundlegende Funktionen, wie eine einfache Suchmaske und eine simpel gestaltete Anzeige von Autoren und Publikationen, an. Es existieren keine weitergehenden Funktionen zur besseren Bedienbarkeit der Website. Zudem verwendet das DBLP ein eher unübersichtliches Design und basiert auf zum Teil veralteten Internettechnologien. Zusammenfassend kann man sagen, dass das DBLP eine konkurrenzlos gute Datenqualität aufweist, jedoch bei der Bedienbarkeit und Benutzerfreundlich-

[2] Stand 1. November 2010

keit Nachholbedarf hat.

Der mühsam erstellte und gepflegte Datenbestand des DBLPs wird von den Betreiber an der Universität Trier in Form einer XML-Datei zur freien Verfügung täglich aktualisiert bereitgestellt. Basierend auf dieser Datei haben die beiden Vorgänger-Projektgruppen den Dortmunder Online-Bibliografieservice oCIOBS entwickelt. Ziel war es, von der guten Datenqualität des DBLPs zu profitieren, gleichzeitig aber einen komfortableren Zugang zu den Informationen zu ermöglichen. Neben der Implementation von umfassenderen Suchfunktionen wurden auch übersichtlichere Publikations- und Autorenseiten erstellt, deren Erscheinungsbild deutlich moderner gestaltet wurde. Publikationen lassen sich zusätzlich in einem Merkzettel speichern. Die grafische Darstellung von Koautoren-Netzwerken und die Möglichkeit diese zur Navigation zu benutzen sind weitere Funktionen zur Steigerung der Benutzerfreundlichkeit. Ein weiteres Merkmal, das implementiert wurde, ist, dass Publikationen oder Sammlungen von Publikationen als BibTeX-Format exportiert und beispielsweise zum Zitieren verwendet werden können.

Neben dem DBLP und oCIOBS haben sich weitere Suchdienste für wissenschaftliche Publikationen mit neuen Funktionen im Internet etabliert. Diese werden im nachfolgenden Kapitel vorgestellt.

1.3.2. Literatur-Suchdienste

Drei große Alternativen zur Suche von wissenschaftlichen Publiaktionen im Internet sind CiteSeer^x, CiteULike und Google Scholar.

CiteSeer^x [50] ist ein System, bei dem im Gegensatz zum DBLP die Dokumente nicht manuell, sondern automatisch hinzugefügt werden. Das Auffinden und Eintragen neuer Publikationen geschieht mittels Robots und Suchmaschinen und hat den Vorteil, dass mit wenig Aufwand eine große Menge und hohe Aktualität an enthaltenen Daten gewährleistet ist. Weiterhin ist es somit leicht möglich neben der Informatik weitere Wissensgebiete einzubinden. Es bringt jedoch den Nachteil der schlechteren Datenqualität mit sich, denn die automatisch extrahierten Informationen werden nicht von den Betreibern überprüft und gegebenenfalls berichtigt. CiteSeer^x bietet den Benutzern nicht nur Metainformationen zu Publikationen, sondern stellt auch frei zugängliche PDFs kostenfrei zum Download bereit. Zudem werden durch Einführung von *MyCiteSeer^x* kollaborative Benutzerfunktionen, wie Tagging angeboten (vgl. Abbildung 1.1 auf Seite 7).

CiteULike [51] ist ein Social-Bookmarking Web-Service, bei dem Bookmarks, also Lesezeichen zu wissenschaftlichen Artikeln und Publikationen aus dem Internet gesammelt und kategorisiert werden. Die Datenerfassung geschieht automatisiert durch Robots, aber auch manuell durch die Benutzer. CiteULike hängt damit sehr stark von der Aktivität der Benutzer ab und bietet daher auch dementsprechend passende Funktionen an. Registrierten Benutzern ist es möglich, neue Bookmarks zum System hinzuzufügen. Stammen diese aus einer der zahlreichen unterstütz-

ten Onlinequellen, werden die bibliografischen Informationen automatisch extrahiert (andernfalls kann eine manuelle Erfassung der nötigen Informationen erfolgen). Anschließend können die Publikationen mit weiteren Informationen angereichert werden, dazu gehören Bewertungen, Notizen und Tags. Über die Tags ist es zudem möglich andere Benutzer mit ähnlichen Interessengebieten zu finden. Des Weiteren können Benutzer für sie interessante Bookmarks zu Bibliotheken zusammenfassen und anderen Benutzern zur Verfügung stellen. Der Vergleich der eigenen Bibliothek mit der Bibliothek anderer Benutzer ermöglicht zusätzlich das Auffinden neuer relevanter Bookmarks. Alle, den Benutzern zur Verfügung stehenden Funktionen, werden im Folgenden aufgelistet.

- Aufruf der eigenen Bibliothek (wahlweise gefiltert nach den ungelesenen Dokumenten) und deren Durchsuchung.
- Anzeige aller Autoren oder Tags in der Benutzer-Bibliothek.
- Export aller bibliografischen Informationen (BibTeX, RIS, PDF, RTF, Text).
- Anzeige des Profils, das neben allgemeinen Angaben auch die zuletzt ausgeführten Aktivitäten in CiteULike auflistet.
- Anzeige der Veröffentlichungen des Benutzers.
- Gruppenzugehörigkeit: Benutzer können sich zu Gruppen zusammenschließen, um ihre Daten zu teilen.
- Anzeige der Nachbarn, d.h. der Personen, welche die gleichen Artikel markiert haben.

Google Scholar [55] ist ein Suchdienst, der ebenfalls speziell der Recherche von wissenschaftlichen Artikeln dient und in Zusammenarbeit von Google mit verschiedenen Fachverlagen entstand. Neben der Suche nach Artikeln und Publikationen ermöglicht Google Scholar auch die Suche nach Zitaten und die Volltextsuche in kostenpflichtigen Dokumenten. Gegebenenfalls zum Download verfügbare PDFs werden direkt beim Suchergebnis angezeigt. Der Suchdienst von Google hat den mit Abstand größten Datenbestand, ohne das genaue Zahlen bekannt sind. Dies ist aber auch der größte Nachteil, denn durch die automatische Erfassung sind viele Metaangaben zu den Suchtreffern falsch oder unvollständig. Neben der mächtigen Suche und einer E-Mail-Benachrichtigung zu neuen Treffern werden keine weiteren Benutzerfunktionen angeboten.

Anhand der drei oben vorgestellten Suchdiensten und der Zusammenfassung dieser (in Abbildung 1.1 auf der nächsten Seite), erkennt man den seit einigen Jahren anhaltender Trend hin zu mehr Benutzerinteraktion und „sozialen Komponenten“ im Internet.

	DBLP	CiteSeer^x	Google Scholar	CiteULike
Datenqualität	Sehr gut	Gut	Befriedigend	Gut
Fachgebiete	Informatik	Hauptsächlich Informatik	Alle (angestrebt)	Nicht festgelegt
Umfang	> 1,4 Mio.	> 1,6 Mio.	>> 10 Mio.	> 4,2 Mio.
Kollaborative Inhalte:				
Tags	✘	✓	✘	✓
Kommentare	✘	✘	✘	✓
Bewertungen	✘	✘	✘	✓

Abbildung 1.1.: Übersicht Alternativsysteme

Zu den grundlegenden Funktionen zählt dabei die Vergabe von Schlagworten (*Tags*) für Publikationen, um diese zu Klassifizieren bzw. näher zu beschreiben. Außerdem ist es, wie am Beispiel CiteULike und CiteSeer^x gezeigt, möglich über die Tags neue relevante Publikationen oder Personen zu finden. Auch der Austausch über die, im Internet gefundenen Informationen nimmt immer stärker zu. Das Kommentieren von Informationen, wie es ebenfalls bei CiteULike möglich ist, generiert neue Daten, die eine Publikation oder einen Artikel beschreiben und somit anderen Benutzern einen zusätzlichen Mehrwert liefern. Die dritte gängige Eigenschaft ist das Bewerten von Literatur. Dies ermöglicht den Benutzern schnell einzuschätzen, ob eine Information relevant ist oder nicht.

Die Gesamtheit dieser Funktionen erlaubt es den Benutzern, neue und für sie relevante Publikationen einfacher zu finden und deren Qualität schnell einschätzen zu können.

Das DBLP besticht zwar durch ihre hohe Datenqualität, bietet jedoch keine sozialen Benutzerfunktionen an. Das gleiche gilt für oCOBS, das auf den Daten des DBLPs basiert. Ziel dieser PG ist also, die hohe Datenqualität des DBLPs mit kollaborativen Inhalten zu vereinen, um den Benutzern das Auffinden der möglichst besten Publikationen zu ermöglichen. Eine genaue Beschreibung, der aus dieser Idee resultierenden Ziele folgt im nächsten Kapitel.

1.4. Ziele

Idee dieser Projektgruppe ist es, die in Abschnitt 1.3.2 auf Seite 5 beschriebenen kollaborativen Benutzerfunktionen mit der herausragenden Datenqualität des DBLPs

zu vereinen um den Benutzern das Verwalten und vor allem das Finden von relevanten Publikationen zu erleichtern. Die im vorherigen Kapitel beschriebenen Alternativsysteme bieten zum Teil kollaborative Funktionen an, haben aber nur eine deutlich geringere Datenqualität im Vergleich zum DBLP. Die Projektgruppe hat sich daher entschieden, das bisherige oDOBS um die nützlichsten kollaborativen Eigenschaften zu erweitern und sich folgende Ziele gesetzt:

- Umsetzung einer Funktion zur Anreicherung von Publikationen um Metadaten, inklusive Kommentarfunktion, Bewertung und Tagging.
- Verbesserung der Suchfunktion, mindestens durch die Möglichkeit der Abspeicherung / des Abonnierens von Suchanfragen.
- Umsetzung von Analysen auf dem angereicherten Datenbestand, mindestens die Ermittlung von Nachbarschaften und die Bewertung der Relevanz von Publikationen.
- Erweiterung von oDOBS um eine Benutzerverwaltung mit persönlichem Profil für jeden Benutzer.

Im Folgenden werden diese Ideen näher beschrieben. Konkrete Umsetzungen werden in Teil III dieses Berichts näher erläutert.

Metadaten für Publikationen

Um den Benutzern von oDOBS einen Mehrwert an Informationen generieren zu können, sollen die Publikationen um folgende Eigenschaften, zur Anreicherung der Informationen um Metadaten, erweitert werden:

- **Kommentare:**
  Zu jeder Publikation kann ein Benutzer Kommentare vergeben, die auf der zugehörigen Publikationsseite angezeigt werden. Jeder Kommentar wird eindeutig mit einem oDOBS-Benutzer verknüpft, was anderen Benutzern ermöglicht das öffentliche Profil des Verfassers des Kommentars einzusehen. Weiterhin kann der Benutzer eigene Kommentare bearbeiten oder löschen. Geschachtelte Kommentare, wie in Diskussionsforen, sind nicht möglich. Es ist aber möglich unpassende oder beleidigende Kommentare an den Administrator von oDOBS zu melden. Personen oder Autoren können nicht kommentiert werden.
- **Bewertung:**
  Registrierte Benutzer haben die Möglichkeit Publikationen zu bewerten. Dieses geschieht in 5 Stufen, von 1 für *schlecht* bis 5 für *sehr gut*. Die 5 Stufen werden in Form von 5 grauen Sternen auf der jeweiligen Publikationsseite angezeigt,

nach der Bewertung werden die Sterne bis zur eigenen Bewertung gelb dargestellt. Die persönliche Bewertung ist jederzeit änderbar. Eine zusätzliche Reihe von 5 Sternen zeigt die durchschnittliche Bewertung der Publikation an. Der aktuelle Bewertungsschnitt wird außerdem noch als Dezimalzahl neben diesen Sternen angezeigt.

- **Tagging:**
  Jeder registrierte Benutzer kann Publikationen taggen, indem er einer Publikation ein oder mehrere Schlagwort(e) (so genannte Tags) hinzufügt. Zur Hilfe werden ihm Tags der anderen Benutzer zur jeweiligen Publikation als Vorschläge angezeigt. Tags werden, wie auch Kommentare, dem jeweiligen Ersteller zugeordnet. Außerdem können falsche Tags gemeldet werden, die dann durch einen Administrator überprüft und gegebenenfalls gelöscht werden. Alle bisher selbst verfassten Tags werden dem Benutzer zusätzlich in seinem Profil dargestellt.

Abonnements

Ein registrierter oCOBS-Benutzer kann seine individuellen Suchanfragen als Abonnements speichern. Anhand dieser Abonnements sollen neue relevante Publikationen für einen Benutzer gefunden werden. Konkret bedeutet dies, dass jeder Benutzer alle neuen, für ihn unbekannte Publikationen in Form einer Liste angezeigt bekommt. Das heißt, dass jedes Mal wenn die gespeicherte Suchanfrage neue Treffer findet, diese dem Benutzer empfohlen werden.

Nachbarschaften

Den Benutzern wird, neben der normalen Suchfunktion, mittels Tag-Nachbarschaften eine weitere Möglichkeit gegeben, für sie unbekannte aber dennoch relevante Publikationen zu finden. Dies geschieht über *Tag-Clouds*, die auf den oben eingeführten Tags basieren und diese grafisch Darstellen. Publikationen sind genau dann benachbart, wenn sie mindestens ein gemeinsames Tag besitzen. Wie auch bei den Koautoren-Netzwerken, wo es möglich ist untereinander bekannte oder zusammenarbeitende Autoren zu ermitteln, ist es durch Navigieren durch die Tag-Clouds möglich zu neuen Publikationen zu gelangen.

Benutzerverwaltung

Zur Realisierung der zuvor genannten Ziele, müssen Benutzer in oCOBS unterschieden werden können. Auf der einen Seite normale Besucher und auf der anderen Seite mitarbeitende Benutzer. Um die mitarbeitenden Benutzer eindeutig identifizieren zu

können, ist eine Benutzerverwaltung unumgänglich. Diese ermöglicht es Kommentare, Tags oder Bewertungen eindeutig zu bestimmten Benutzern zuzuordnen. Die folgende Auflistung zeigt die Funktionalitäten, welche die Benutzerverwaltung in oCOBS aufweisen muss, um anschließend weitere Funktionen, wie das Anreichern der Publikationen um Metadaten, darauf aufbauen zu können:

- **Registrierung und Login:**
  Ein Internetnutzer kann sich bei oCOBS registrieren und einloggen. Nach der Registrierung wird ein persönliches Profil angelegt, welches jederzeit nach einem Login eingesehen und bearbeitet werden kann.

- **Öffentliches Profil:**
  Das Profil eines Benutzers hat einen öffentlichen Bereich, welcher von anderen registrierten oCOBS-Benutzern eingesehen werden kann. Dazu gehören unter anderem Informationen zur Person wie z.B. E-Mail-Adresse, Name oder eine Beschreibung des Benutzers. Außerdem können dort anderen Benutzern eigene Interessengebiete über die persönliche Tag-Cloud (vgl. Kapitel 8.3 auf Seite 53) zugänglich gemacht werden. Alle Angaben im öffentlichen Profil sind freiwillig und können jederzeit geändert bzw. entfernt werden.

- **Einstellungen / Profil ändern:**
  Der Benutzer kann verschiedene profilbezogene Einstellungen ändern. Dazu gehört unter anderem das öffentliche Profil, die E-Mail-Adresse u.s.w. Außerdem kann er sein Profil bei oCOBS löschen.

- **Sperren und Löschen von Benutzern:**
  Der Administrator von oCOBS kann bestimmte Benutzer sperren und eine Liste mit gesperrten Benutzern verwalten. In einem weiteren Schritt kann ein Benutzer auch dauerhaft aus oCOBS entfernt werden.

1.5. Übersicht

Im zweiten Abschnitt wird das bestehende System analysiert, um existierende Schwachstellen zu identifizieren. Zunächst wird auf die Architektur von oCOBS und dessen technischen Aufbau eingegangen. Weiterhin wird ein Überblick über die uns vorliegende Dokumentation gegeben, welche danach bewertet wird. Daraufhin werden die einzelnen Bausteine des Systems analysiert. Dies wären das Klassenmodell, die Persistenzschicht (*Kapitel 3 auf Seite 20*), der Adminclient (*Kapitel 6 auf Seite 36*), der Webclient (*Kapitel 4 auf Seite 27*) und der Datenimport (*Kapitel 7 auf Seite 40*).Die Kapitel zum Webclient und dem Adminclient befassen sich hier mit den Funktionalitäten der jeweiligen Komponenten. Außerdem wird ein Stabilitätsproblem analysiert, welches im Zusammenhang mit der Suche aufgetreten ist (*Kapitel 5 auf Seite 33*).

Der dritte Abschnitt beschreibt die Weiterentwicklungen im Rahmen der *Projektgruppe 540: Semantic* oCIOBS. Hierzu gehören unter anderem die verschiedenen möglichen Realisierungen der Benutzerverwaltung (*Kapitel 9*), die Anreicherung des Datenbestandes durch Metadaten (*Kapitel 8.3*), die Verbesserungen am Datenimport (*Kapitel 10*) und die Änderungen an der Suchfunktion zur Lösung des bestehenden Stabilitätsproblems (*Kapitel 11*). Es werden jeweils Technologien vorgestellt, welche zur Realisierung der entsprechenden Ziele in Frage kommen. Diese werden miteinander verglichen, um Vor- und Nachteile gegeneinander abzuwägen.

Abschließend wird die bisherige Arbeit zusammengefasst und ein Ausblick für fehlende aber nützliche Funktionalitäten gegeben.

Teil II.

Systemanalyse

2. Architektur

2.1. Java EE

Die *Java Enterprise Edition* (Java EE) [23] setzt auf der Java Standard Edition auf und erweitert diese um zusätzliche APIs, um Anwendungen implementieren zu können, welche eine Client-Server-Architektur aufweisen. Im Gegensatz dazu dient die *Java Standard Edition* (Java SE) [24] zur Entwicklung von Anwendungen für Einzelsysteme und stellt hierfür viele verschiedene Standardschnittstellen (APIs) bereit. Ein herausstechendes Merkmal ist die *Java Virtual Machine* (JVM), welche Code zur Laufzeit interpretiert, wodurch die erzeugte Anwendung auf jedem System lauffähig ist, für das die JVM implementiert ist. Zusätzlich bietet JSE eine große Runtime-Bibliothek, in der viele Klassen vorimplementiert sind, was für den Entwickler häufig erheblichen Minderaufwand bedeutet. Java EE enthält sowohl die JVM, als auch die Java Runtime-Bibliothek.

Um die Performance einer verteilten Anwendung mit potentiell vielen gleichzeitigen Anfragen sicherzustellen, verfügt Java EE über eine sehr gute Skalierbarkeit, für die insbesondere *Enterprise Java Beans* (EJB), eine der neu hinzugekommenen APIs, verantwortlich ist. Diese sind Java-Klassen, welche auf einen bestimmten Geschäftsprozess in einem System ausgelegt sind und somit für eine Trennung von Präsentationslogik und Geschäftslogik sorgen. Durch Instance-Pooling wird sichergestellt, dass die Anzahl der bereitgestellten EJBs je nach Bedarf erhöht oder verringert wird. Diese Verschiedenen Bean-Instanzen arbeiten voneinander unabhängig, weshalb ein Fehler in einer dieser Instanzen nicht dazu führt, dass ein Fehler im kompletten System entsteht.

Weiterhin sind Komponenten einer Java EE Anwendung modular entwickelt und garantieren eine hohe Wiederverwendbarkeit und Robustheit.

Sämtliche Komponenten einer Java EE Anwendung lassen sich per Hot-Deployment zur Laufzeit aktualisieren, wodurch zu jeder Zeit ein Zugriff auf die Komponente möglich ist und damit eine hohe Verfügbarkeit garantiert wird.

2.1.1. Schichtenmodell

Für eine gute Skalierbarkeit bei Java EE sorgt auch die Trennung von logischen Ebenen, welche sich gut in einem Schichtenmodell darstellen lassen. Es wird hier zwischen drei verschiedenen Schichten unterschieden (siehe Abbildung 2.1 auf der

nächsten Seite).

Die oberste Schicht ist die Darstellungsschicht auf der sich Anwendungen befinden, welche erzeugte Inhalte präsentieren. In den häufigsten Fällen handelt es sich um Webbrowser, die serverseitig dynamisch erzeugten Html-Code darstellen. Hier befinden sich allerdings auch Client-Anwendungen.

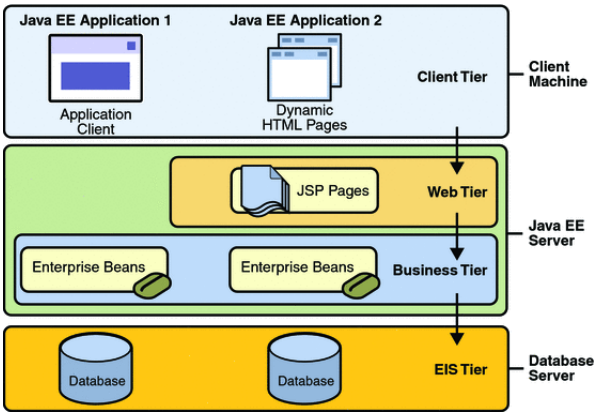

Abbildung 2.1.: Java EE Schichtenmodell

Die mittlere Schicht befindet sich auf einem Application-Server und lässt sich wiederum in Webinhalte und Geschäftslogik aufteilen. Die Webinhalte befinden sich in einem Webcontainer. Hierbei kann es sich zum Beispiel um *Java Servlets*, *Java Server Pages* und *Java Server Faces* handeln.

Ein *EJB-Container* kann mehrere Enterprise Java Beans enthalten und dient diesen als Laufzeitumgebung und überwacht ihre Ausführung. Er ist sowohl für das Instance-Pooling, als auch für das Lifecycle Handling zuständig. Ersteres sorgt dafür, dass genügend Bean-Instanzen für eingehende Anfragen vorhanden sind, während letzteres unter anderem dafür verantwortlich ist, dass Instanzen von alten EJBs, sobald sie nicht mehr benötigt werden, von neuen Versionen ersetzt werden.

Es existieren zwei verschiedene Arten von EJBs. *Session Beans* sind nichtpersistente serverseitige Komponenten, in denen die Logik eines Geschäftsprozesses bzw. eines Anwendungsfalls oder eines Dienstes implementiert ist. [7] Es existieren zwei verschiedene Arten von Session Beans. Zum einen gibt es die *Stateful Session Beans* (SFSB). Diese können vom aufrufende Client übergebene Informationen über mehrere Methodenaufrufe hinweg speichern. Ein SFSB bleibt einem Client so lange zugeordnet, bis es von diesem wieder freigegeben wird. Speichert eine Session Bean keinerlei clientspezifischen Daten, so spricht man von einer *Stateless Session Bean* (SLSB). Eine SLSB wird einem aufrufenden Client nur für einen Methodenaufruf zugewiesen. *Message Driven Beans* (MDB) sind Objekte, die zur asynchronen Kommunikation dienen.

Damit bestimmte Daten in einem System dauerhaft gespeichert werden können, und auch bei einem möglichen Neustart oder Absturz weiterhin vorhanden sind,

müssen sie persistiert werden. Daten, die im System persistent gehalten werden sollen, werden in *Persistent Entities* erfasst. Häufig wird zur Speicherung persistierter Daten eine relationale Datenbank verwendet. Zwischen den Persistent Entitys und der Datenbank findet ein *Object Relational Mapping* statt, welches die Persistent Entities auf Datenbanktabellen abbildet. Dieser Vorgang wird von der *Java Persistence API* (kurz JPA) ausgeführt.

Die unterste Schicht ist die Datenschicht, welche eine oder mehrere Datenbanken enthält und zur Ablage der gerade erwähnten persistenten Daten dient. Die Kommunikation erfolgt über die *Java Database Connectivity* (JDBC), welche durch Connection-Pooling einen effizienten Zugriff auf die gewünschten Daten bietet.

2.1.2. JBoss

Um eine Webanwendung funktionsfähig zu machen, bedarf es Servern, auf denen diese Anwendungen bereitgestellt werden. Diese werden als Application Server bezeichnet und stellen Dienste und Funktionen für Webentwickler zur Verfügung, welche wiederum über standardisierte Software-Schnittstellen angesprochen werden können.

Der Application Server wird der Geschäftslogik-Ebene zugeordnet und bildet das Bindeglied zwischen der Präsentationsschicht und der Persistenzschicht. Da diese beiden Schichten auf verschiedenen Serverinstanzen betrieben werden können, muss eine zusätzliche Technologie die Kommunikation der beiden Schichten gewährleisten. Für diesen Zweck werden bei oDOBS Webservices verwendet (siehe 2.2 auf Seite 18).

Unter dem Begriff Application Server werden oft die Geschäftslogik-Ebene und die Kommunikations-Ebene zusammengefasst, wobei eigentlich zwei Server beteiligt sind: der Application Container und der Web Container. Im Falle des JBoss, welcher ein solcher Application Server ist, übernimmt Apache Tomcat die Aufgabe des Web Containers [56].

Verwendung in der Projektgruppe

Derzeit wird JBoss in der Version 4.0.5 als Umgebung der oDOBS Hauptanwendung benutzt. Eine der Öffentlichkeit zugängliche Live-Instanz der stabilen Version des gesamten Projektes läuft auf einer Server-Hardware in einer Instanz des JBoss [1]. Die Suchfunktionen und das Datenmodell liegen im Application Container und die Servlets des Webclients befinden sich in einem als JBoss Service gestarteten Tomcat. Das Adminbackend wird ebenfalls von der JBoss Instanz zur Verfügung gestellt.

Zusätzlich existierte eine Test-Instanz von oDOBS, die zur Weiterentwicklung verwendet wurde. Auf dieser wurden Integritätstests und neue Import-Algorithmen der DBLP ausprobiert. [2] Nach neueren Implementierungen in oDOBS, musste diese Tes-

[1] `http://www.odobs.de`
[2] `http://pavo.cs.uni-dortmund.de:8080/odobs/`

tinstanz, aufgrund von Kompatibilitätsproblemen auf eine Debian Instanz verschoben werden.

2.2. oDOBS

Die Architektur von oDOBS ist in verschiedene Komponenten gegliedert (siehe Abbildung 2.2). Diese Komponenten lassen sich dem Schichtenmodell der J2EE Plattform in verschiedenen Ebenen ansiedeln. Die einzelnen Komponenten werden in den folgenden Kapiteln dokumentiert.

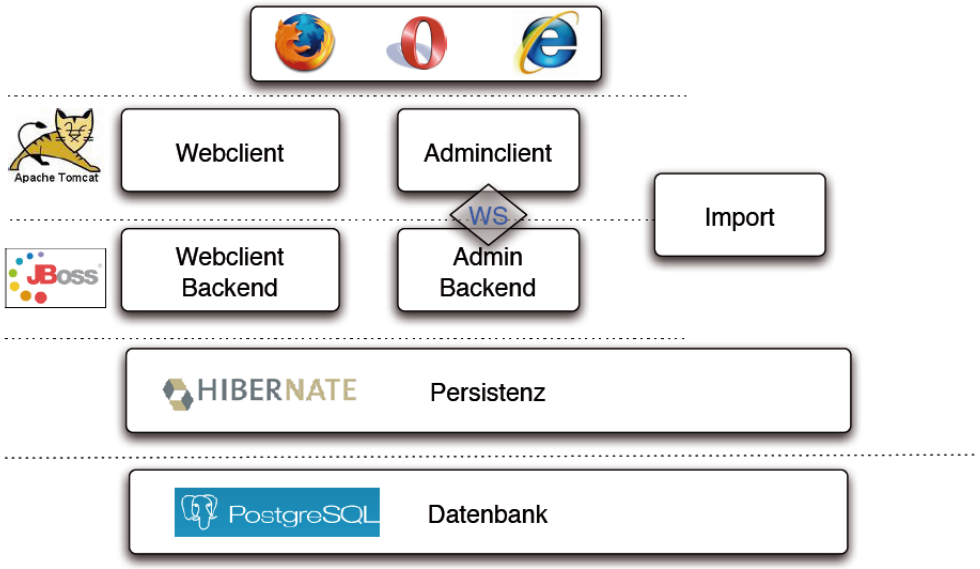

Abbildung 2.2.: Architektur und Komponenten [4]

Die Grundlage des Systems bietet die *Datenbankschicht*, in der das Datenmodell angesiedelt ist. Die Zugriffe der Anwendung auf das Datenmodell erfolgen durch die vermittelnde *Persistenzschicht*. oDOBS verfügt über zwei verschiedene Benutzerschnittstellen, die in den nächsten beiden Abschnitten vorgestellt werden: Der Webclient bietet zusammen mit dem dazugehörigen Backend die Suchfunktionalität für den Endnutzer an. Der Adminclient erlaubt den administrativen Zugriff auf oDOBS und dient als Frontend für die im Admin-Backend durch einen Web Service angebotenen Verwaltungsfunktionen. Auf beide Schnittstellen kann der Nutzer mittels eines Web-Browsers zugreifen, da sowohl der Webclient als auch der Adminclient als Web-Anwendungen implementiert sind. Zusätzlich gibt es noch den Import-Mechanismus, welcher die DBLP-Daten in die oDOBS Datenbank überführt.

Dieser wird jedoch in einem separaten Kapitel beschrieben (siehe Kapitel 7 auf Seite 40).

Webclient

Der Webclient ist die Benutzerschnittstelle, über die der Endnutzer mit dem Dienst interagieren kann. Dem Nutzer werden dort viele unterschiedliche Funktionalitäten geboten, die eine umfangreiche Literaturrecherche ermöglichen. Diese Funktionalitäten werden in Kapitel 4 auf Seite 27 ausführlich beschrieben. Der Webclient wird durch die Technologien *JavaServer Pages* und *Servlets* [2] realisert.

Adminclient

Der Adminclient stellt im Gegensatz zum Webclient die Benutzerschnittstelle für die Administratoren von oCIOBS zur Verfügung. Diese haben dort diverse Möglichkeiten, um den Datenbestand von oCIOBS zu pflegen. Zwei Beispiele dafür sind die Doublettensuche und das manuelle Editieren von Publikationsdaten. Zusätzlich können einzelnen Administratoren unterschiedliche Tasks (siehe auch 6.1 auf Seite 38) mit verschieden hohen Prioritäten zugewiesen werden. Diese Tasks werden entweder automatisch durch das System, wie zum Beispiel durch zu überprüfende Doubletten der Doublettensuche, oder durch Nutzer des Webclients, die Fehler über Feedbackformulare melden können, erzeugt. Die genaue Funktionsfülle wird in Kapitel 6 auf Seite 36 erläutert. Die von der PG 540 neu hinzugefügten Funktionen finden sich in Kapitel 12.1 auf Seite 85. Im Gegensatz zum Webclient kommen im Adminclient statt Java Server Pages als Technologie Java Server Faces zum Einsatz. Dieser Unterschied ist dem viel geringerem Nutzeraufkommen im Adminclient geschuldet, sodass die Vorteile der nicht so performanten Java Server Faces genutzt werden können. Neben den Java Server Faces werden im Adminclient *Webservices* genutzt, welche die Schnittstelle zwischen dem Web- und Adminclient bilden.

Webservices

Ein Webservice ist ein Service, der über ein Netzwerk, meistens das Internet, verfügbar ist und einen XML-basierten Nachrichtenaustausch zwischen den Teilnehmern ermöglicht. Die Teilnehmer sind meist Softwaresysteme, die so plattformunabhängig miteinander kommunizieren können.

Webservices bauen auf offenen Internetstandards auf und benutzen bekannte Internetprotokolle, wie zum Beispiel die Benutzung des HTTP zum Transport oder die Verwendung von URIs zur eindeutigen Identifizierung des Services. Die Webservices zielen hauptsächlich auf die Kommunikation von einzelnen Maschinen ab, im Gegensatz zu Webseiten oder Web-Anwendungen, die für Kommunikation zwischen

Mensch und Maschine geschaffen wurden. Sie besitzen keine grafische Benutzeroberfläche. Des Weiteren können Webservices als Ressource für Anwendungen und andere Webservices dienen oder diese aufrufen. Webservices sind modular und sich selbst beschreibend. Das bedeutet, dass der Webservice zu jeder Zeit weiß, welche Funktionen er erfüllen kann, welche Eingaben dafür benötigt werden und zu welcher Ausgabe diese führen. Diese Informationen stellt er anderen Webservices oder potenziellen Nutzern zur Verfügung. Webservices sind transparenter als Web-Anwendungen, da der aktuelle Zustand immer gelesen werden kann. Webservices sind lose gekoppelt, das bedeutet, dass der einzelne Webservice keine Information über das Verhalten oder die Implementierung der Anwendung oder des Services, mit dem er interagiert benötigt. Dies führt zu einer sehr hohen Flexibilität. [14]

Verwendung in oDOBS

Bei oDOBS werden alle Verwaltungsfunktionen dem Adminclient vom Adminbackend durch Webservices zur Verfügung gestellt. Konkret betrifft dies die Suchfunktion und die Kommunikation zwischen Adminclient und -backend. Webservices stellen eine sehr einfache Möglichkeit dar, um Daten und Funktionen über ein Netz zu kommunizieren. Bei oDOBS sind durch die Nutzung von Webservices der Adminclient und Webclient unabhängig und müssen nicht lokal auf derselben Maschine betrieben werden. Dadurch wird ein gewisses Maß an Flexibilität gewonnen. Sie sind plattformunabhänig, erfordern wenig Implementierungsaufwand und sind deshalb eine gute Lösung, um Funktionen, wie die Adminfunktionen von oDOBS, zu realisieren. Die starke Standardisierung durch Protokolle und XML ist ein weiteres großes Plus.

3. Datenmodell & Persistenz

Um dem riesigen Datenbestand der DBLP, den oCOBS in strukturierter Weise beinhalten soll, gerecht zu werden, wurde schon von der ersten Projektgruppe ein abstrakt gehaltenes Datenmodell entworfen. Hinter diesem Datenmodell, bestehend aus Java-Klassen, verbirgt sich eine Umsetzung der Klassen auf Tabellen in einer Datenbank. Als Vermittlungs- oder auch Persistenzschicht zwischen den Java-Klassen und der Datenbank findet das Software-Framework Hibernate (siehe Kapitel 3.2 auf Seite 25) in der Version 3.2.0 Anwendung.

3.1. Datenmodell

Das Datenmodell von oCOBS lässt sich im Wesentlichen in drei Bereiche gruppieren:

- Klassen für den bibliografischen Datenbestand
- Klassen für das Logging
- Klassen für das Task-Management

3.1.1. Bibliografie

Zweck des bibliografischen Datenbestandes von oCOBS ist die Speicherung der Daten, die die DBLP zur Verfügung stellt. Dies sind größtenteils Literaturangaben zu Publikationen aus z.B. Fachzeitschriften, Konferenzen, aber auch ganze Bücher oder Doktorarbeiten.

Bibliografie Klassen

Wesentliche Klassen zum Speichern der Bibliografie sind deshalb:

- `Category`: Mittels dieser rekursiven Datenstruktur werden in oCOBS Kategorien gespeichert
- `Publication`: In dieser Klasse werden die eigentlichen Publikationen gespeichert
- `Person`: Diese Klasse beschreibt einen Autor und/oder Editor

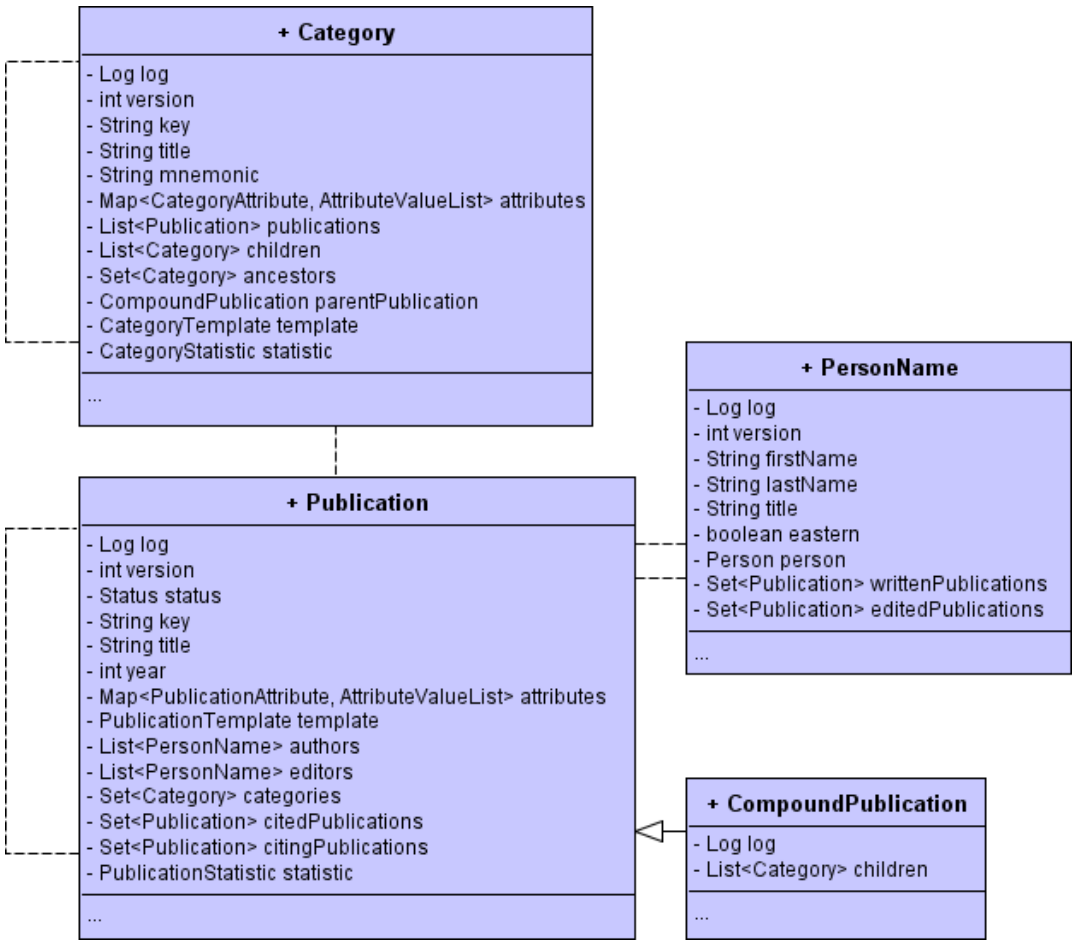

Abbildung 3.1.: Wesentliche Bibiliographie-Klassen

Die Klasse `Category` hat die Aufgabe, die Verzeichnisstruktur der DBLP nachzubilden. Dazu besitzt sie eine Liste von Kind-Kategorien und eine Liste der Publikationen, die in der aktuellen Kategorie enthalten sind. Zugleich wird auch noch eine Menge von Vorgänger-Kategorien gespeichert. Eine jede Kategorie verfügt zusätzlich noch über weitere Eigenschaften wie z.B. einen Titel oder einen Status.

Ebenfalls eine Liste von Kategorien besitzt die Klasse `Publication`. Eine Publikation kann somit in mehreren Kategorien vertreten sein. Jede einzelne Publikation ist weiterhin in der Lage, beliebig viele andere Publikationen zu zitieren, sowie von diesen zitiert zu werden. Da es möglich ist, dass z.B. an einem Buch mehrere Autoren geschrieben haben, verfügt jede Publikation über eine Liste von Autoren und des Weiteren auch über eine Liste von Editoren, welche beide durch die Klasse `Person` dargestellt werden (siehe Abbildung 3.4 auf Seite 24) [12, S.8].

Die Tatsache, dass die Literaturangaben zu diesen unterschiedlichen Publikationstypen in ihrem Informationsgehalt meist sehr voneinander abweichen, machte es

erforderlich, ein sehr flexibles Datenmodell zu erstellen. Den drei Klassen `Category`, `Publication` und `Person` ist es deshalb gemeinsam, dass an sie beliebige Attribute geknüpft werden können. Dies geschieht über die Klassen `CategoryAttribute`, `PublicationAttribute` und `PersonAttribute`, die alle von der gemeinsamen Klasse Attribute abgeleitet sind (siehe Abbildung 3.2) [12, S.5f].

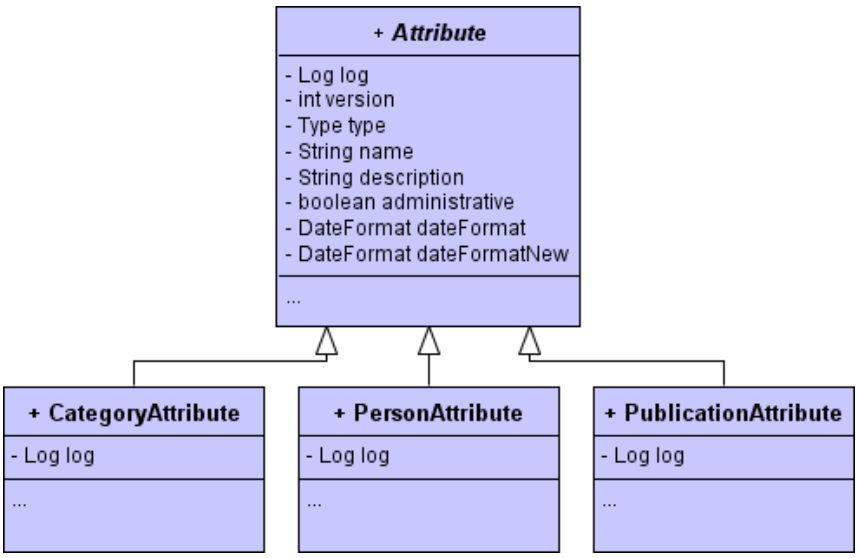

Abbildung 3.2.: Attribute-Klassen

Um den verschiedenen Publikationstypen der DBLP gerecht zu werden, wurde für die Klassen `Category` und `Publication` die Möglichkeit geschaffen, Vorlagen für wiederkehrende Publikationstypen zu erstellen. Diesem Zweck dienen die Klassen `CategoryTemplate` und `PublicationTemplate`, die beide von der Klasse `Template` abgeleitet wurden (siehe Abbildung 3.3 auf der nächsten Seite).

Jede Vorlage beinhaltet dabei eine Menge von benötigten und eine Menge von optionalen Attributen. Die meisten in der Praxis eingesetzten Vorlagen entsprechen dabei den verschiedenen Typen des BibTeX-Formats wie z.B. article, book und conference. Die dabei verwendeten Attribute gleichen dann jeweils denen der entsprechenden BibTeX-Typen. Mögliche Beispiele hierfür sind Eigenschaften wie Titel und Veröffentlichungsjahr einer Publikation [12, S.7].

Zur Speicherung der Autoren und Editoren einer Publikation dient in oCOBS die Klasse `PersonName`. Da es in der Praxis vorkommt, dass Autoren unter mehreren Namen bzw. Pseudonymen veröffentlichen, oder sich ihr Name ändert, bietet das Datenmodell die Möglichkeit, mehrere Instanzen der Klasse `PersonName` unter eine Person zusammenzufassen. Dies geschieht in der gleichnamigen Klasse `Person` (siehe Abbildung 3.4 auf Seite 24) [12, S.7].

Weiterhin können für die drei bekannten Klassen `Category`, `Publication` und `Person` statistische Werte gespeichert werden. Dies geschieht über die Klassen

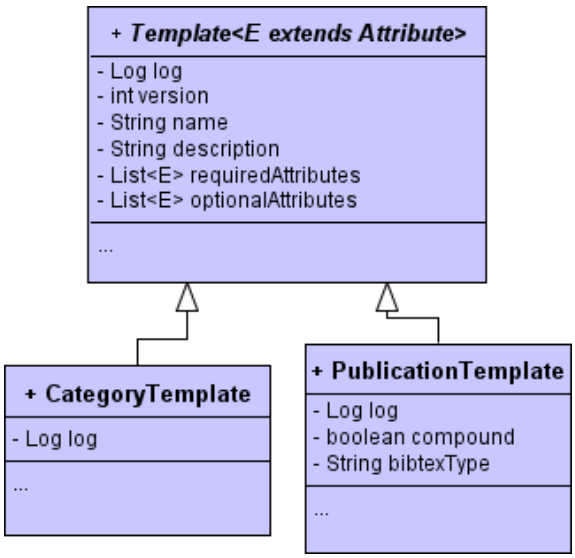

Abbildung 3.3.: Template-Klassen

`CategoryStatistic`, `PublicationStatistic` und `PersonStatistic` (siehe Abbildung 3.5 auf Seite 25).

3.1.2. Logging

Im Bereich des Adminclients bietet oCOBS eine sehr detaillierte Protokollierung aller Änderungen am bibliografischen Datenbestand. Einen zentralen Punkt spielt dabei die Benutzer- und Rollenverwaltung. Eine Rolle (Klasse `Role`) verfügt dabei über eine Menge von vorgegebenen Rechten. Einzelne Benutzer (Klasse `User`) sind wiederum einer oder mehreren Rollen zugeordnet, aus denen sich dann die effektiven Rechte des Benutzers ergeben (siehe Abbildung 3.6 auf Seite 25).

Beim Logging wird die Benutzerkennung zusammen mit weiteren Daten protokolliert. Diese Daten beinhalten dann, je nach Ereignis, z.B. Informationen über geänderte Publikationen, Autoren oder Kategorien. Des Weiteren werden auch Änderungen an Benutzerkonten und Benutzerrechten protokolliert.

Eine genaue Beschreibung ist den Java-Klassen beginnend mit den Präfixen `AbstractLogEntry` und `LogEntry` zu entnehmen.

3.1.3. Task-Management

Für den Fall, dass ein Webclient-Benutzer eine Meldung an die Administratoren von oCOBS schicken will, gibt es ein Feedback-System. Mit diesem können Benutzer sowohl Anmerkungen bezüglich einer Publikation oder eines Autors als auch ganz generelle Nachrichten an die Administratoren schicken. Dieses Feedback-System wie-

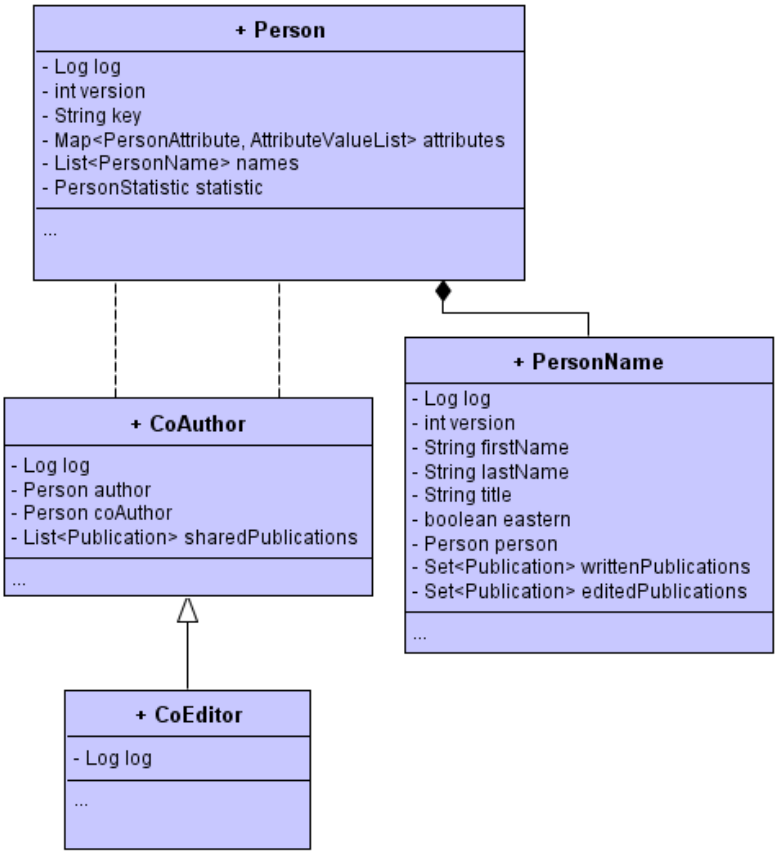

Abbildung 3.4.: Klassen bzgl. Autoren / Editoren

derum baut auf einem allgemeinen Task-Management-System auf, das jede anfallende Aufgabe (Klasse `Task`) einem Administrator zuweist (siehe Abbildung 3.7 auf Seite 26). Jede Aufgabe verfügt dabei über einen Titel, eine Beschreibung, einen Status und je nach Ursprung auch noch über weitere Angaben wie z.B. die E-Mail-Adresse des Benutzers, der eine Meldung verschickt.

Neben der Verwendung für das Feedback-System findet die Task-Management-Datenstruktur noch Anwendung bei der Doublettensuche. Hier besteht die Aufgabe darin, eine mögliche, automatisch-ermittelte Doublette, durch einen Administrator auswerten zu lassen.

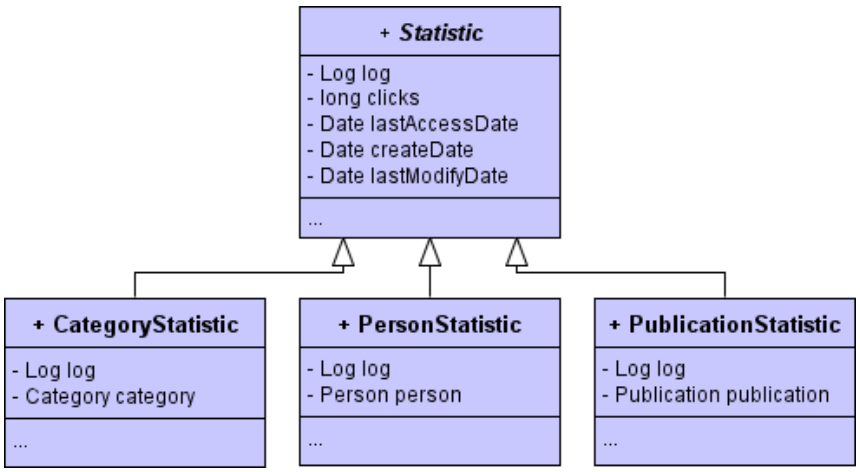

Abbildung 3.5.: Statistic-Klassen

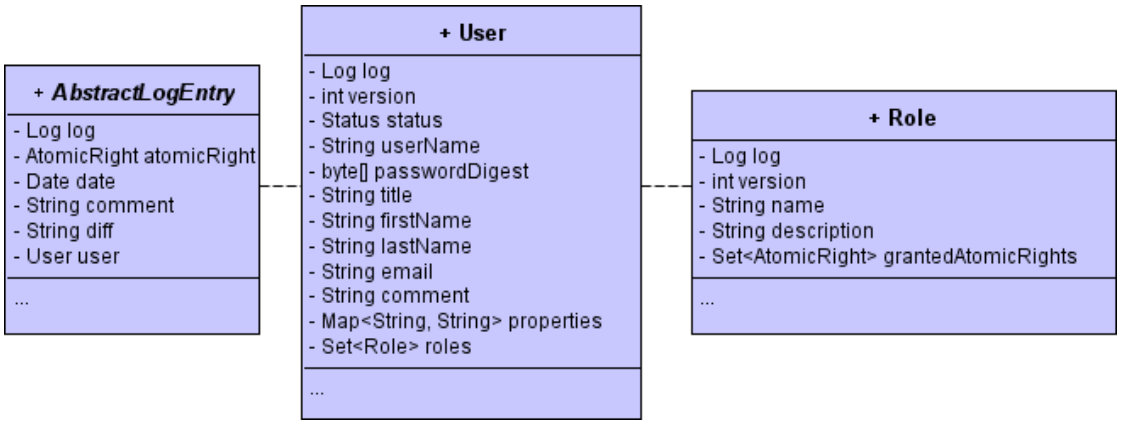

Abbildung 3.6.: Logging

3.2. Hibernate

Als Persistenzmechanismus verwendet oCIOBS Hibernate [42]. Dabei handelt es sich um ein Framework für *Object-Relational-Mapping* unter Java. Unter Object-Relational-Mapping wird eine Abstraktionsebene verstanden, die dem Programmierer die Arbeit abnimmt, sich mit einer konkreten Datenbanktechnologie und deren Anfragesprache auseinanderzusetzen. Stattdessen können die beim Programmieren anfallenden Instanzen von Objekten in einer relationalen Datenbank gespeichert und aus dieser auch wieder geladen werden.

Der Programmierer kann den gewohnten Komfort der objektorientierten Programmierung ausnutzen, während z.B. Änderungen an den Attributen eines Objektes nahezu transparent in der Datenbank geltend gemacht werden. Auch beim Laden der Daten aus einer Datenbank kann dies mittels objektorientierter Programmierung

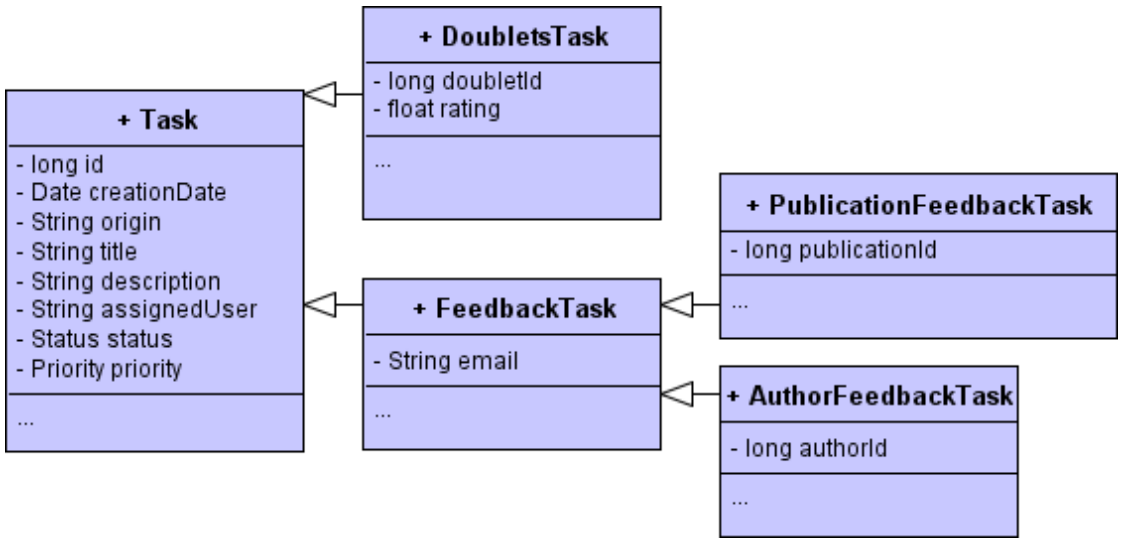

Abbildung 3.7.: Task-Management

geschehen. Hierfür hält Hibernate die Criteria-API bereit.

Darüber hinaus besteht auch noch die Möglichkeit, mittels eines verallgemeinerten SQL-Dialekts, HQL genannt, unabhängig von der verwendeten Datenbanktechnologie Anfragen zu stellen. Alternativ können natürlich auch wie gewohnt direkt SQL-Anfragen des entsprechenden Datenbankmanagementsystems verwenden werden, eine Option, die gerade bei performancekritischen Abfragen eine Rolle spielen kann und an einigen Stellen auch bei oCOBS genutzt wurde.

Damit Hibernate weiß, auf welche Art und Weise die benutzten Objekte auf die Tabellen und Spalten einer Datenbank abgebildet werden sollen, muss dieser Zusammenhang durch eine Zuordnung verständlich gemacht werden. Diese Zuordnung, auch Mapping genannt, kann entweder durch eine XML-Datei oder durch Java-Annotations geschehen. Bei oCOBS fiel die Wahl auf Letzteres.

4. Webclient

Eine der wichtigsten Komponenten von oDOBS ist der Webclient. Er beinhaltet sämtliche Benutzerfunktionen und dient als direkte Schnittstelle für die Literatur suchenden oDOBS-Nutzer. Das folgende Kapitel beschreibt die einzelnen Funktionen von oDOBS aus der Sicht der Benutzer.

4.1. Layout & Startseite

Das Layout von oDOBS ist generell in drei Bereiche geteilt: Eine obere und eine seitliche Menüleiste sowie mittig der Inhalt.

Die obere Menüleiste beinhaltet die Hauptnavigation, welche es den Benutzern ermöglicht jederzeit auf die Startseite, die Bibliografie, die Suche, den Merkzettel und die Hilfe zuzugreifen. Die Hauptnavigation ist von jeder oDOBS-Seite erreichbar und sieht immer gleich aus.

Die seitliche Menüleiste passt sich hingegen dynamisch dem Inhalt der jeweiligen Seite an. Meistens befinden sich dort inhaltsbezogene Optionen sowie ein Hilfe-Text.

Der Bereich zur Darstellung des Inhalts nimmt naheliegenderweise den meisten Platz in Anspruch. In diesem Bereich werden, wie auch bei der seitlichen Menüleiste, die Informationen in Blöcken gruppiert angezeigt, die durch ihre hervorgehobenen Überschriften und farblichen Abgrenzungen deutlich als solche zu erkennen sind. [8, S.66].

Die Startseite dient dem Anwender als Einstiegspunkt für oDOBS. Auf dieser hat er die Möglichkeit, sich mittels der schon genannten Menüleisten, durch oDOBS zu navigieren. Oben und links angebrachten Navigationsleisten helfen ihm, sich für einen der weiterführenden Bereiche zu entscheiden.

Im Inhaltsbereich wird dem Benutzer kurz und präzise erklärt, was oDOBS ist. Außerdem hat er hier schon die Möglichkeit die Suche zu benutzen.

Zusätzlich wird auf der Startseite noch eine kleine Statistik anstelle des Hilfstextes angezeigt. Diese beinhaltet die Anzahl der Publikationen und der Autoren in der Datenbank und zeigt die Anzahl aktiver Benutzern, die oDOBS momentan verwenden, an. [8, S.61].

4.2. Bibliografie

Da oCOBS über 1,4 Mio. Publikationen verwaltet [1], werden diese den Benutzern, zur besseren Übersicht, strukturiert angeboten. So ist der bibliografische Datenbestand in oCOBS hierarchisch angeordnet und wird so dem Benutzer präsentiert.

Die Bibliografie umfasst dabei eine Vielzahl verschiedenster Publikationstypen, wie z.B. Konferenzbände, Artikel in einem Konferenzband, Monografien, Bücher, Zeitschriften, Diplom- und Doktorarbeiten (siehe 7.1.1.1 auf Seite 41).

Folgt der Benutzer, auf der Startseite, dem Link Bibliografie (siehe Abbildung 4.1), so gelangt er zur entsprechenden Seite, auf der nach den unterschiedlichen Publikationstypen differenziert wird. Auf der obersten Ebene sieht der Benutzer dabei nur die übergeordneten Publikationskategorien, wie z.B. Publikationen in Fachzeitschriften und Konferenzbänden, aber noch keine konkreten Publikationen [8, S.64] [8, S.7ff].

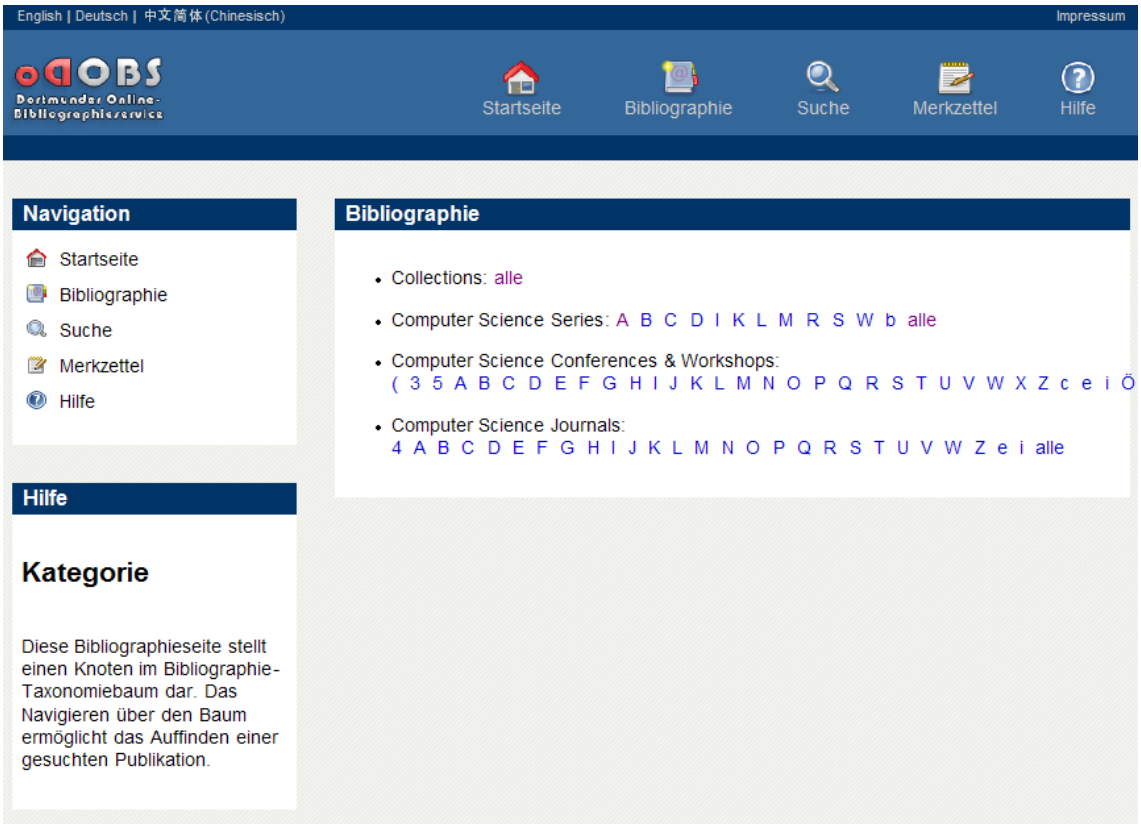

Abbildung 4.1.: Darstellung der Bibliografieseite [32]

All diese verschiedenen Publikationstypen werden in oCOBS durch das Konzept von Kategorien realisiert. In diesen Kategorien werden Publikationen nach bestimm-

[1] Stand 1. November 2010

ten Kriterien, wie z.B. Konferenzbände oder Zeitschriftenartikel hierarisch zusammengefasst. Innerhalb dieser Kategorien sind die jeweiligen Publikationen dann noch weiter in Unterkategorien aufgeteilt, z.B. alphabetisch nach Publikations- oder Konferenzname. Dies ermöglicht den Benutzern das Auffinden von Publikationen, ohne deren konkrete Namen oder Autoren zu kennen.

Sind zu einer Kategorie in der Datenbank weitere Informationen verfügbar, z.B. weiterführende Links, so werden diese auf der entsprechenden Seite ebenfalls angezeigt. Die linke Menüleiste beinhaltet zudem eine Liste mit übergeordneten Kategorien, in denen die aktuelle Kategorie enthalten ist [8, S.64] [8, S.7ff].

Hat der Benutzer sich über die Bibliografie oder die Suche zu einer Detailseite einer Publikation begeben, so werden auf dieser die Details der Publikation angezeigt. Zu diesen gehören u.a. der Titel des Werkes und das Veröffentlichungsjahr. Angegeben wird auch eine Liste der Autoren sowie der Koautoren und Editoren. Über diese kann der Benutzer direkt zu den entsprechenden Personen gelangen. Existieren in der Datenbank Referenzen von oder zu anderen Publikationen, so werden diese ebenfalls dargestellt.

In der linken Menüleiste besteht die Option, die aktuelle Publikation zu einem Merkzettel hinzuzufügen. Außerdem wird hier die Gesamtzahl der Einträge des Merkzettels angezeigt. Weiterhin besteht die Möglichkeit, die aktuelle Publikation in Form einer Literaturangabe zu exportieren. Die unterstützten Formate sind: PDF, HTML und BibTeX. Zusätzlich werden noch die Kategorien angezeigt, denen die momentane Publikation zugeordnet ist. [8, S.63]

Publikationen können mehreren Autoren und Editoren zugehörig sein, wobei ein Autor in seiner klassischen Rolle einen Mitverfasser/-urheber meint (siehe Kapitel 1.1 auf Seite 2). Ein Editor dagegen meint einen Herausgeber, ihm obliegt u.a. eine gewisse Sorgfaltspflicht bzgl. der Güte einer Publikation.

So finden sich auf der Seite eines Autors oder Editors (siehe Abbildung 4.2 auf der nächsten Seite), sofern vorhanden, Listen der verfassten Publikationen, Koautoren, editierten Publikationen sowie Koeditoren. Auch hier ist es wieder möglich, zu den entsprechenden Autorenseiten zu gelangen.

Zusätzlich gelangt der Benutzer über die Menüleiste zur Visualisierung des Koautoren- und Koeditorennetzwerkes (siehe Abbildung 4.3 auf Seite 31).

Weiterhin werden zusätzliche Informationen aus der Datenbank angezeigt, wenn diese dem Autor / Editor zugeordnet sind. Dies kann z.B. ein Link auf die Homepage der Person sein [8, S.61f].

Zu jedem Autor existiert ein Koautoren- und ein Koeditorennetzwerk. In diesem (siehe Abbildung 4.3 auf Seite 31) werden jeweils die Koautoren oder die Koeditoren mithilfe eines Graphen visualisiert. Um den Namen des Autors verteilen sich kreisförmig die Namen der Koautoren oder Koeditoren. Eine Kantenbeschriftung gibt Aufschluss darüber, wie oft die entsprechenden Personen miteinander publiziert oder editiert haben. Mittels eines Doppelklicks auf einen der Namen kann der Benutzer sich durch den Graphen navigieren. So ist es möglich einen guten Überblick

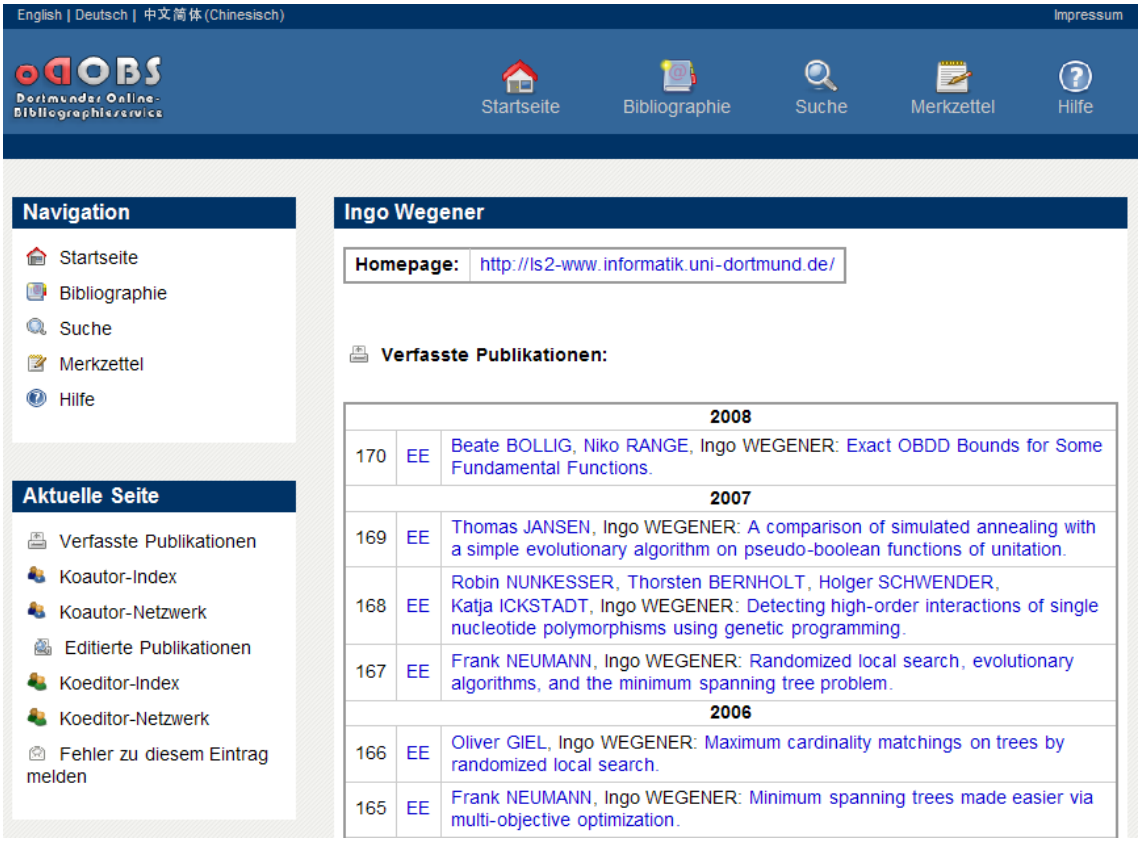

Abbildung 4.2.: Darstellung einer Autorenseite [32]

über das Kontakt-Netzwerk einer Person zu bekommen [4, S.82].

4.3. Suche

In den meisten Fällen wird der Benutzer die Suche einer manuellen Recherche im oDOBS Datenbestand vorziehen. Die Suchseite bietet dem Benutzer dafür eine einfache und eine erweiterte Suche, wobei standardmäßig die einfache Suche angezeigt wird. Jede der beiden Sucharten bietet eine Einstellung an, die die Anzahl angezeigter Ergebnisse festlegt.

Bei der einfachen Suche können lediglich einige wenige Einstellungen getroffen werden. Neben der Angabe des Suchstrings kann ausgewählt werden ob als Ergebnis die gefundenen Publikationen oder die gefundenen Autoren angezeigt werden. Zusätzlich zu der Suche zeigt oDOBS dem Benutzer mögliche Verbesserungsvorschläge in Form von alternativen Suchbegriffen an. Diese werden anhand eines Ähnlichkeitsmaßes ermittelt [8, S.64].

Verwendet der Benutzer die erweiterte Suche, so können bis zu drei Autoren mit-

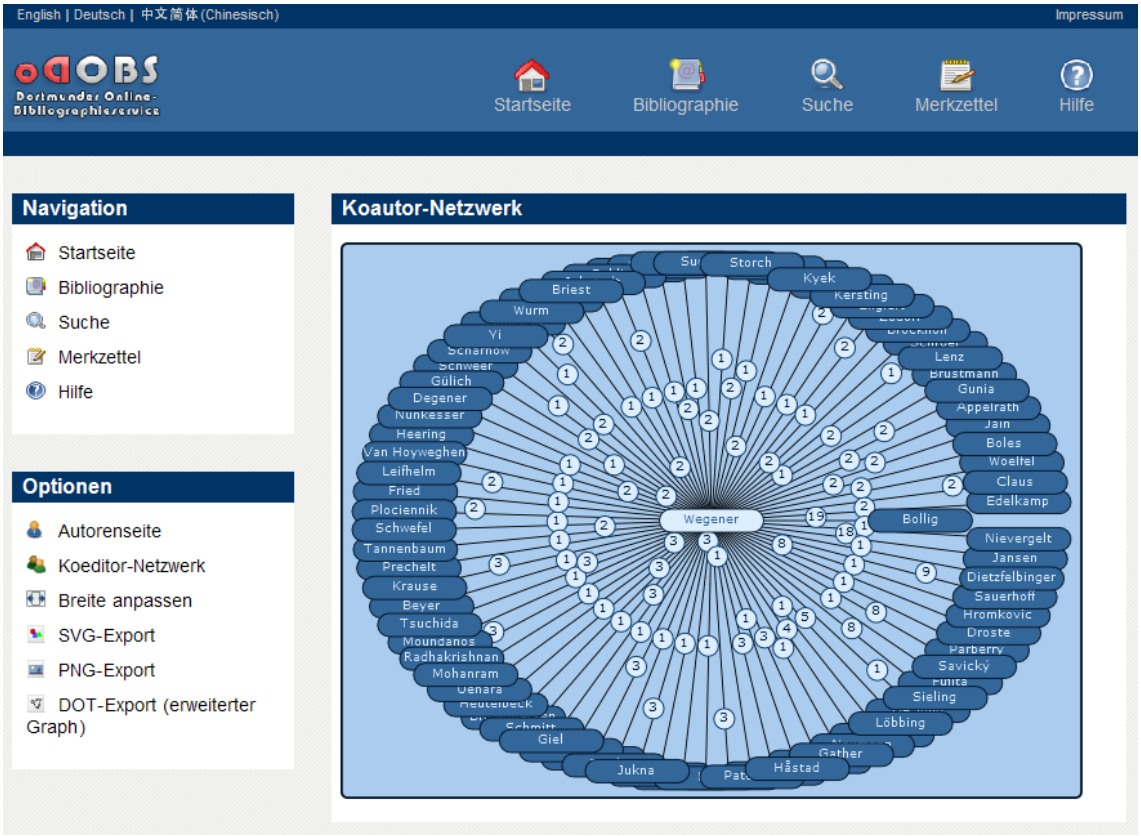

Abbildung 4.3.: Darstellung eines Koautoren-Netzwerk [32]

tels ihrer Vor- und Nachnamen als Kriterium angegeben werden. Um die Suchergebnisse auch bzgl. Publikationen einschränken zu können, können der Titel und die Kategorie, sowie ein Zeitfenster für die Veröffentlichung angegeben werden.

Die Ergebnisse der Suche lassen sich nach verschiedenen Kriterien, wie z.B. Titel, Jahr oder Ranking sortieren. Darüber hinaus kann sich der Benutzer hier ebenfalls Publikationen merken oder als Literaturangabe exportieren [8, S.64].

Erfahrenen Benutzern bietet die Suchmaske der einfachen Suche die Möglichkeit ihre Anfragen detaillierter, mittels einer komplexen Syntax, zu stellen. Diese bietet eine größere Anzahl an Möglichkeiten, um die Suche noch gezielter einzuschränken. Hier kann dann genauer beschrieben werden, in welchen Attributen der Benutzer suchen möchte. Ein Attribut kann dabei z.B. der Vorname des Autors oder der Titel der Publikation sein [8, S.76ff].

4.4. Merkzettel, Hilfe und Feedback-Formular

Gerade für das wissenschaftliche Arbeiten ist es notwendig korrekt zu zitieren, auch hierfür bietet oCOBS eine Funktion. Publikationen können zu einem Merkzettel hinzugefügt oder von diesem wieder entfernt werden. Der Merkzettel kann dann, wenn eine Liste von Literaturangabe benötigt wird, im BibTeX-Format exportiert werden. Auch hierbei hilft oCOBS dem Benutzer. So befinden sich auf der Seite des Merkzettels die vom Benutzer gemerkten Publikationen. Hier können einzelne oder alle der Publikationen des Merkzettels gelöscht, oder exportiert werden. Sinnvoll kann dies z.B. für das Erstellen eines Literaturverzeichnisses sein [8, S.64]. Weiterhin verbirgt sich unter dem Menüpunkt Hilfe eine Liste von Hilfe-Themen. Wählt der Benutzer eines dieser Themen aus, so erhält er zu diesem weitere Informationen [8, S.64]. Außerdem besteht die Möglichkeit über das Impressum, innerhalb der Hilfe oder über das Seitenmenü einer Publikation zu einer Feedback-Seite zu gelangen. Auf dieser kann unter optionaler Angabe einer E-Mail-Adresse eine Nachricht an die Administratoren gesendet werden, um z.B. auf einen Fehler hinzuweisen [4, S.80].

5. Systemstabilität

Die Suche ist die wohl am häufigsten verwendete Funktion in oCOBS und damit auch die, bei der sich am ehesten Schwachstellen in der Implementierung feststellen lassen. Im Regelbetrieb konnte beobachtet werden, dass in regelmäßigen Abständen Abstürze aufgrund fehlenden Speichers auftraten. Daher lag die Vermutung nahe, dass die Implementierung von oCOBS ein Speicherleck enthält, das dafür sorgt, dass im laufenden Betrieb der zur Verfügung stehende Speicher voll läuft und somit zum Absturz der Webapplikation führt.

5.1. Werkzeug

Speicherlecks äußern sich im Programmbetrieb dadurch, dass Objektinstanzen, die angelegt werden, nicht wieder freigegeben werden. Dadurch belegen diese Objektinstanzen Speicher, der nicht mehr genutzt werden kann. Gerade bei längerem Programmbetrieb führt dies dazu, dass der belegte Speicher das Programm in seiner Ausführung behindert und sogar zu Abstürzen führen kann. Normalerweise erledigt der Garbage Collector diese Aufgabe, indem alle nicht mehr referenzierten Objekte automatisch aus dem Speicher entfernt werden. Bei Speicherlecks kann der Garbage Collector nicht reagieren, da noch Referenzen auf die nicht mehr benötigten Objekte existieren. Dies kann beispielsweise geschehen, indem Referenzen durch Programmierfehler in Listen, oder ähnlichen Datenstrukturen abgelegt werden, ohne zu beachten, dass diese wieder entfernt werden müssen, sobald das jeweilige Objekt nicht mehr benötigt wird. Um feststellen zu können, welche Objekte instanziiert, aber nie wieder freigegeben werden, wird ein Tool benötigt, das im laufenden Programmbetrieb auslesen kann, wie viele Instanzen welcher Klassen vorhanden sind. Für diese Arbeit sind die sogenannten Profiler geeignet. Diese sind für die Programmanalyse gedacht und verfügen über die Fähigkeit, die gewünschten Informationen zu ermitteln.

Für den JBoss-Server gibt es den JBossProfiler [25], der sich als Webapplikation in den JBoss eingliedern lässt. Da in diesem Fall noch die JBoss Version 4.05 verwendet wird, musste auf die Version 1 des JBossProfilers zurückgegriffen werden, die als Einzige zu den älteren JBoss-Versionen kompatibel ist. Mit Hilfe der JMX-Konsole des JBoss ist es nach erfolgter Installation des JBossProfilers möglich, die Methode *heapSnapshot()* aufzurufen. Nach diesem Aufruf erstellt der Profiler im bin-Verzeichnis des JBoss drei Dateien, die den sogenannten Snapshot darstellen.

Die Datei *_classes* enthält eine Auflistung aller im Speicher befindlichen Klassen. In der Datei *_objects* werden die Instanzen der in *_classes* aufgelisteten Klassen zusammengefasst. Eine weitere Datei namens *_references* enthält Informationen darüber, welche Objekte sich untereinander referenzieren. Für die in diesem Abschnitt durchgeführten Tests wurden lediglich die Dateien *_classes* und *_objects* verwendet. Darüber hinaus stellt der JBossProfiler eine Funktion *forceGC()* zur Verfügung, die es ermöglicht, den Java Garbage Collector manuell auszulösen. Dadurch werden alle nicht mehr referenzierten Objekte aus dem Speicher entfernt.

5.2. Suche des Speicherlecks

Nach der Installation und Konfiguration des JBossProfilers wurde folgendes Testszenario eingerichtet, das die Nutzung der Suchfunktion durch mehrere Benutzer simulieren sollte. Hierzu wurde die automatisierte Ladefunktion des Opera Webbrowsers genutzt, indem auf fünf Registerkarten zur gleichen Zeit alle fünf Sekunden unterschiedliche Suchanfragen gestartet wurden. Alle fünf Minuten wurde während des Testbetriebs mit dem JBossProfiler der Garbage Collector manuell angestoßen und ein Snapshot erstellt, der die benötigten Informationen der existierenden Instanzen bestimmter Klassen enthielt. Nach einem Testdurchlauf, der 40 Minuten dauerte, wurden die Daten mithilfe eines selbst erstellten Tools analysiert und daraufhin verglichen. Bei dieser Analyse stellte sich, wie im Listing 5.1 auf der nächsten Seite zu sehen ist, heraus, dass die Instanzen, der bei jeder Suchanfrage erstellten Log-Einträge (*ExpressionLog* und *LogEntryWebClientQuery*) im Speicher verbleiben und offensichtlich nicht vom Garbage-Collector erkannt und beseitigt werden.

5.3. Analyse des Quellcodes

Nach erfolgter Analyse mithilfe des Profilers bestand der nächste Schritt im Auffinden der betroffenen Stelle im Quelltext. Hierbei lag das Hauptaugenmerk auf der Verwendung und dem Verbleib der angesprochenen Log-Einträge. Erste Anlaufpunkte waren in diesem Schritt die beiden Methoden *evaluateQueryLog()* und *evaluate()*, in denen die Instanzen der Log-Einträge erstellt werden. Nach dem einfachen Auskommentieren dieser Methoden konnte beobachtet werden, dass das Speicherleck nicht mehr auftritt. Also konnte im Zuge dieser Aktion die Quelle des Lecks schon einmal grob eingekreist werden. Da die erstellten Objekte nach dem Persistieren nicht weiter genutzt werden, lag die Vermutung nahe, dass sich der Grund für das Speicherleck im *PersistenceManager* befindet. Diese Vermutung ließ sich untermauern, indem ausschließlich der Aufruf der *persist*-Methode auskommentiert wurde. Nach dieser Auskommentierung ließ sich das Speicherleck nicht mehr antreffen. Dies bestätigte die zuvor formulierte Vermutung und stellte klar, dass der Aufruf

```
Snapshot direkt nach dem Start des JBoss:
066 x "de/odobs/backend/security/AtomicRight"

Snapshot nach 5 Minuten:
452 x "de/odobs/literature/PersonStatistics"
298 x "de/odobs/backend/logging/ExpressionLog"
298 x "de/odobs/backend/logging/LogEntryWebClientQuery"

Snapshot nach 10 Minuten:
669 x "de/odobs/literature/PersonStatistics"
581 x "de/odobs/backend/logging/ExpressionLog"
581 x "de/odobs/backend/logging/LogEntryWebClientQuery"

Snapshot nach 15 Minuten:
870 x "de/odobs/backend/logging/ExpressionLog"
870 x "de/odobs/backend/logging/LogEntryWebClientQuery"
669 x "de/odobs/literature/PersonStatistics"

Snapshot nach 20 Minuten:
1170 x "de/odobs/backend/logging/ExpressionLog"
1170 x "de/odobs/backend/logging/LogEntryWebClientQuery"
669 x "de/odobs/literature/PersonStatistics"

Snapshot nach 40 Minuten:
2055 x "de/odobs/backend/logging/ExpressionLog"
2055 x "de/odobs/backend/logging/LogEntryWebClientQuery"
669 x "de/odobs/literature/PersonStatistics"
```

Listing 5.1: Testergebnisse

der *persist*-Methode die Objekte persistiert, diese jedoch nicht freigibt, sodass der Garbage-Collector sie nicht als freizugebende Objekte erkennt.

Nach der Identifizierung des Speicherlecks mussten im nächsten Schritt Änderungen am Quelltext vorgenommen werden, die dafür sorgen, dass belegter Speicher zuverlässig wieder freigegeben wird. Dies wurde im Zuge der Anpassung der Suchfunktion durchgeführt und wird in Kapitel 11.1 auf Seite 80 näher erläutert.

6. Adminclient

Neben der Schnittstelle zur Literatursuche, verfügt oCIOBS auch über eine Administrationsschnittstelle. Mit Hilfe dieser können Administratoren auf die Datenbestände, wie Publikationen, Kategorien und Autoren zugreifen und diese verändern bzw. verwalten. Außerdem lassen sich Administrator-Konten erstellen und Rechte vergeben. Die Schnittstelle ist eine Web-Anwendung, welche mithilfe des Java ServerFaces Frameworks, kurz `JSF` [43], implementiert wurde und als Frontend für Administratoren dient. Sie greift auf die Webservice-Schnittstelle des Admin-Backends zu und nutzt dessen angebotene Funktionen. Das Admin-Backend besitzt selber keine Benutzeroberfläche, bietet Nutzern aber alle nötigen Funktionen über *Webservices* an.

6.1. Aufbau des Adminclients

Zugang zum Client

Der Adminclient ist als Web-Applikation implementiert worden und kann dadurch mit jedem regulären Internet-Browser über eine Internetadresse erreicht werden. Damit sichergestellt ist, dass nur Benutzer mit den nötigen Rechten oCIOBS administrieren, wird vor dem Betreten der Administrationsoberfläche ein Benutzername mit Passwort abgefragt. Außerdem haben die Administratoren hier die Möglichkeit zwischen den Sprachen `Deutsch` und `Englisch` zu wählen, da oCIOBS als mehrsprachige Plattform konzipiert ist. Nach erfolgreichem Login gelangt ein Admin auf die Verwaltungsoberfläche, auf der mittels Reitern durch die Funktionen navigieren kann.

Kategorien

Alle Publikationen werden in oCIOBS zur bessern Übersicht in Kategorien, wie zum Beispiel `Computer Science Series`, einsortiert. Unter dem Reiter `Kategorien` ist es möglich, diese zu durchsuchen, neu anzulegen oder deren Zugehörigkeit zu ändern. Es ist immer möglich, neue Unterkategorien von Kategorien zu erstellen oder die Zugehörigkeit von Unterkategorien zu ändern. Kategorien können auch umbenannt oder gelöscht werden.

Publikationen

Ähnlich wie bei Kategorien kann im Reiter Publikationen der gesamte Datenbestand an Publikationen durchsucht und verwaltet werden. Unter `Publikationen suchen` hat der Administrator die Möglichkeit, über eine einfache Suchfunktion nach Publikationen zu suchen. Ein Admin hat zwei Möglichkeiten auf Publikationen zuzugreifen. Zum Einen über eine einfache Suchfunktion und zum Anderen durch Auswahl einer der zuletzt bearbeiteten Publikationen. Neben der Verwaltung der Publikationen hat ein Admin die Möglichkeit manuell Neue zu erstellen, dazu muss ihr ein Titel gegeben, ein Veröffentlichungsjahr zugeteilt und eine entsprechende Vorlage gewählt werden. Anschließend lassen sich publikationsspezifische Attribute, wie zum Beispiel die Autoren oder die Seitenzahl, hinzufügen. Außerdem können die neuen Publikationen in Kategorien eingeordnet werden. Möchte man neue publikationsspezifische Attribute hinzufügen oder Vorhandene bearbeiten, so ist dies im „Attribut-Pool“ möglich. Es gibt beispielsweise das Attribut ISBN-Nummer. Diese lassen sich dann zu Vorlagen für Publikationen zusammenfassen und abspeichern. Späteres bearbeiten dieser ist jederzeit möglich.

Personen

In diesem Abschnitt des Adminclients hat man die Option nach bestehenden Autoren zu suchen, Neue zu erstellen oder Vorhandene zu bearbeiten. Für das Erstellen eines neuen Autors muss der Name eintragen und zusätzlich vermerkt werden, ob dieser aus dem asiatischen Raum kommt. Optional kann eine Homepage hinzugefügt werden. Wenn ein Autor als asiatisch gekennzeichnet ist, bedeutet das, dass die Reihenfolge von Vor- und Nachname variabel ist. Dies ist notwendig, da bei asiatischen Namen oft unterschiedliche Schreibweisen für den selben Autor in den Publikationen existieren.

Benutzerverwaltung

In der Benutzerverwaltung hat ein Administrator die Möglichkeit, nach bestehenden Benutzern zu suchen oder Neue zu erstellen. Es können auch neue Rollen generiert werden, in welchen verschiedene administrative Rechte festgelegt sind. In diesem Bereich werden nur Nutzer der Admin-Oberfläche und nicht des Webclients verwaltet. Einem neu erstellten Nutzer müssen Attribute wie Name und Rolle zugewiesen werden, dies muss geschehen, damit sichergestellt wird, dass dieser nur den ihm erlaubten Tätigkeiten nachgehen kann. Bei den Rollen gibt es derzeit nur die des vollwertigen Administrators und die des Besuchers. Zudem muss bestimmt werden, auf welche Funktionen die jeweiligen Rollen zugreifen dürfen. Es können jederzeit alte Benutzerprofile verändert und ihnen neue Rollen zugewiesen werden.

Aufgaben

Weiterhin besteht die Möglichkeit für Administratoren Aufgaben, sogenannte *Tasks*, zu verwalten und abzuarbeiten. Hier kann nachgesehen werden, welche Tasks existieren und ggfs. noch zu bearbeiten sind. Tasks sind generische Objekte und bilden administrative Aufgaben ab. Diese Objekte können sich auf Ressourcen im Datenbestand beziehen und einem Administrator zur Bearbeitung zugewiesen werden. Sie werden an verschiedenen Punkten im System erzeugt und zeigen dabei zum Beispiel Konflikte beim Import von Daten an oder leiten Rückmeldungen von Benutzern des Webclients an einen Administrator weiter. Tasks besitzen eine Priorität und einen Lebenszyklus oder auch Status, damit die Administratoren die Abarbeitung der Aufgaben besser organisieren können.

Statistiken

In diesem Abschnitt ist es möglich sich alle Statistiken zu den in oDOBS enthaltenen Komponenten anzusehen. Die Statistiken lassen sich durch eine Suche filtern. Dabei können die bestehenden Kategorien, Publikationen, Personen, Benutzer, Webclient und Dubletten betrachtet werden. Unter `Webclient` wird angezeigt, wie viele Suchen in den letzten Stunden, Tagen und Wochen durchgeführt wurden und unter Dubletten wann wie viele und welche gefunden wurden.

Dubletten

Die Dublettensuche wird prinzipiell nicht von oDOBS selber ausgeführt, sondern als externes Skript gestartet. Die Dublettensuche ist, wie der Import, ein eigenständiges von der oDOBS-Webapplikation getrenntes Programm. Dieser Bereich dient daher nur zur Statusabfrage und nicht zur Bedienung der Dublettensuche.

Im Dublettenmanager können jedoch Publikationen anhand einer *Blacklist* gefiltert werden. Die Blacklist dient dazu, häufig vorkommende Begriffe anders zu behandeln, beziehungsweise zu ignorieren. Zum Beispiel haben Magazine meistens denselben Namen, jedoch handelt es sich um Ausgaben aus verschiedenen Monaten. Indem der Titel des Magazins in die Blacklist eingetragen wird, kann sichergestellt werden, dass die unterschiedlichen Ausgaben nicht als Dubletten voneinander erkannt werden. Dabei wird nur der Titel der Publikation gefiltert. Es können jederzeit neue Einträge in die Blacklist hinzugefügt und Vorhandene wieder gelöscht werden. Aktuelle Änderungen wirken sich direkt auf laufende Dublettensuchen aus.

DMS

DMS steht für das Dokumentenmanagementsystem in oDOBS. Die entsprechende Funktion ist erst von der letzten Projektgruppe hinzugefügt worden. Sie dient dazu,

Dokumente in ein bestehendes DMS-Repository zu laden oder die Daten anderweitig zu exportieren. Die Dokumente werden selber nicht in oCOBS gespeichert, vielmehr dient das DMS dem Zweck der Verknüpfung von Publikationen mit den dazugehörigen Dokumenten im Internet.

7. Import

Dieses Kapitel soll einen Überblick über die verschiedenen Möglichkeiten verschaffen, die oCOBS bereitstellt, um Publikations- und Kategoriedaten zu importieren.

7.1. Importarten

Wie bereits erwähnt, existieren mehrere Möglichkeiten, um Daten in oCOBS zu importieren. Grundlage für den Datenbestand von oCOBS bilden die Daten der DBLP. Diese sind über die DBLP-Homepage unter [59] zu beziehen. Es werden sowohl Publikationsdaten als auch Kategoriedaten angeboten. Grundsätzlich muss der Publikationsimport vor dem Kategorieimport geschehen.

Weiterhin ist es möglich, Daten aus Dateien im *BibTeX*-Format (siehe Kapitel 7.1.2 auf Seite 42) auszulesen und dem Datenbestand von oCOBS hinzuzufügen. Dies ist für uns speziell von Bedeutung, da am Lehrstuhl 5 der Technischen Universität Dortmund viel mit Publikationsdaten im BibTeX-Format gearbeitet wird und somit einige Publikationsdaten im BibTeX-Format gespeichert sind.

7.1.1. DBLP-Import

Der Schwerpunkt der DBLP liegt auf der möglichst hohen Korrektheit der gespeicherten Informationen. Es wird Wert darauf gelegt, dass die Einträge nicht wie bei anderen Systemen (z.B. CiteSeer^x oder Google Scholar) automatisch durch Robots, sondern manuell erfasst werden. Insbesondere die korrekte Wiedergabe von Personennamen wird angestrebt. Um dies zu erreichen und zur Schonung der knappen Ressourcen haben sich die Betreiber der DBLP gegen eine ausführliche Beschreibung der Publikationen entschieden. Der Aufwand für die manuelle Erfassung ist dennoch so groß, dass pro Tag nur ca. 500 Einträge zur DBLP hinzugefügt werden können.

Um von der hohen Datenqualität zu profitieren (vgl. auch Abschnitt 1.3 auf Seite 3) verwendet oCOBS die von der DBLP bereitgestellten Daten. Es ist also notwendig, die DBLP-Daten in das bei oCOBS verwendete Datenformat zu überführen. Die Abschnitte 7.1.1.1 und 7.1.1.2 auf der nächsten Seite geben eine Übersicht über das bei der DBLP verwendete Datenmodell, Abschnitt 7.3 auf Seite 45 beschreibt die Ausführung des Imports.

7.1.1.1. Publikationsdaten

Zu Beginn des DBLP Projekts wurden die Daten direkt als statische HTML-Seiten gespeichert. Einige Jahre später wurde mit dem Aufkommen des XML-Standards auf diesen umgeschwenkt. Die bibliografischen Daten der gespeicherten Publikationen sind nun in einer einzigen großen XML-Datei gespeichert, der `dblp.xml`[8]. Listing 7.1 auf der nächsten Seite zeigt einen Ausschnitt dieser Datei. Neben der XML-Deklaration und dem Wurzelelement `dblp` enthält die Datei für jede im Datenbestand enthaltene Publikation ein Element, dessen Tag-Name sich von dem Typ der Publikation ableitet. Diese können sein `article`, `inproceedings`, `proceedings`, `book`, `incollection`, `phdthesis`, `mastersthesis` und `www`. Jedes dieser Elemente hat die Attribute `key` und `mdate`. Ersteres ist ein eindeutiger Bezeichner für das Dokument im DBLP Datenbestand. Das zweite Attribut enthält das Datum, an dem die letzte Modifikation des Datensatzes erfolgte. Diese Information wird für die Weiterentwicklung dieser Projektgruppe an der Importfunktion verwendet, um die Geschwindigkeit zu steigern (vgl. Abschnitt 10.1.1 auf Seite 75).

Auf der nächsten Hierarchiestufe befinden sich Elemente, die die bibliografischen Informationen der Publikation beschreiben. Sie orientieren sich an den entsprechenden BibTeX-Feldern (z.B. `author`, `title`, ...), es ist aber nicht festgelegt, welche Eigenschaften tatsächlich angegeben werden. Weitergehende Informationen zum Aufbau der Datei finden sich in [11].

7.1.1.2. Kategoriedaten

Die Zuordnung der Publikationen zu *proceedings* oder *journals* ist weiterhin mittels HTML realisiert. Dabei existiert für jeden Konferenzband/jede Fachzeitschrift eine eigene Seite mit Links zu den enthaltenen Publikationen. Diese Seiten werden jede Nacht aus den sogenannten *bibliography hypertext (BHT)* Dateien erzeugt, die eine HTML ähnliche Syntax aufweisen, die jedoch leicht erweitert wurde. Das DBLP-Team bietet aber die Möglichkeit, auch diese Daten in zusammengefasster Form herunterzuladen. Zu diesem Zweck gibt es eine weitere XML-Datei (`dblp_bht.xml`[9]), die mithilfe leichter Anpassungen an den XML-Standard aus den BHT-Dateien erzeugt wird. In ihr befinden sich die BHT-Dateien in Form von `<bht>...</bht>` Blöcken (vgl. [11]).

Das System als solches kann nicht als optimal bezeichnet werden. Dies sieht auch Michael Ley [11] so, doch es wird bewusst auf Änderungen verzichtet – sofern nicht wichtige Neuerungen damit einhergehen – um die Stabilität nicht zu gefährden. Außerdem ist die fehlende Zeit (bedingt durch die Konzentration auf die Aufnahme neuer Inhalte) ein limitierender Faktor. Daher wird z.B. auch heute noch auf den

[8] Größe ca. 660 MB
[9] Größe ca. 100 MB

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
...
<incollection mdate="2002-01-03" key="books/acm/kim95/Blakeley95">
        <author>Jos&eacute; A. Blakeley</author>
        <title>OQL[C++]: Extending C++ with an Object Query Capability.</title>
        <pages>69-88</pages>
        <booktitle>Modern Database Systems</booktitle>
        <url>db/books/collections/kim95.html#Blakeley95</url>
        <year>1995</year>
</incollection>
...
</dblp>
```

Listing 7.1: Ausschnitt der `dblp.xml`

Einsatz eines Datenbankmanagementsystems verzichtet, obwohl es bereits verschiedene Ansätze für die Einführung gab.

7.1.2. BibTeX-Import

BibTeX ist ein weitverbreiteter Standard um Literaturverzeichnisse in einer einfach strukturierten Textdatei zu speichern. Eine solche BibTeX-Datei hat üblicherweise die Endung `.bib` und muss eine spezielle Syntax erfüllen. Ursprünglich war BibTeX als Literaturverzeichnis für in Latex geschriebene wissenschaftliche Ausarbeitungen gedacht. Somit war es möglich, Quellenangaben und Zitate zu erstellen und diese losgelöst vom eigentlichen Inhalt zu verwalten. Mittlerweile wird BibTeX allerdings auch außerhalb von Latex verwendet, um Literaturverzeichnisse zu verwalten. Viele große Verlagshomepages bieten die Möglichkeit an, Daten in BibTeX exportieren zu lassen.

BibTeX-Syntax

Ein BibTeX-Datum wird durch ein `@` eingeleitet. Darauf folgt die Art des Datums, welches angibt, von welcher Art die verwendete Literatur ist. Ein häufig verwendeter Literaturtyp ist `book`, wodurch die Quelle als Buch definiert wird. Soll auf einen wissenschaftlichen Artikel verweisen werden, so wird `article` verwendet. Der dritte besonders häufig verwendete Literaturtyp ist `misc`. Hierbei handelt es sich um einen generischen Datentyp, mit dem solche Literatur erfasst wird, die keinem der anderen Literaturtypen zugeordnet werden kann. Besonders gebräuchlich ist dies bei Websites.

In geschweiften Klammern folgt nun der eigentliche Literatureintrag, welcher aus einem Schlüsselwort und einer Menge von Tags besteht. Das Schlüsselwort dient als eindeutiger Identifikator für das Datum und erlaubt es, später innerhalb des Latexdokumentes auf dieses zu referenzieren. Bei den Tags handelt es sich um Attribute, die das Dokument genauer beschreiben. Dies können zum Beispiel `author`, `title`,

```
@book{far2005,
        author="Jim Farley and William Crawford",
        title="Java Enterprise in a Nutshell",
        publisher="O'Reilly",
        year="2005",
        edition="3rd"
}

@misc{sun.javatut,
        title="Java EE 5 Tutorial",
        author="Sun Microsystems",
        note="\texttt{http://java.sun.com/javaee/5/docs/tutorial/doc}"
}
```

Listing 7.2: Beispiel einer BibTeX-Datei

`note`, `isbn` oder `year` sein. Literaturtypen haben jeweils eine Menge von Pflichttags und optionalen Tags. Listing 7.2 zeigt, wie eine solche BibTeX-Datei aussehen kann. Weitere Informationen zu BibTeX und eine genauere Auflistung von möglichen Literaturtypen und deren notwendigen und optionalen Tags finden sich in [58].

7.2. Technische Realisierung

Um Daten in oCOBS importieren zu können, wird ein Verfahren benötigt, um diese aus ihrem Ursprungsformat in das oCOBS-eigene zu konvertieren. Diese Konvertierung geschieht, indem ein entsprechender Parser den Ursprungscode einliest und anhand der Semantik entsprechende Datenstrukturen in dem Format, das von oCOBS verwendet wird, erstellt. Hierbei wird für jedes Datenformat ein spezieller Parser benötigt. oCOBS verwendet für den Import von DBLP-Publikations- und DBLP-Kategoriedaten StAX (Streaming API for XML). Für den BibTeX-Import wird Javabib verwendet. Weiterhin wird für den Datenimport das *Java Application Building Center* (JavaABC) verwendet , welches ein Werkzeug zur grafischen Modellierung von Prozessen ist.

JavaABC

Bei der Entwickelung der Importfunktion von oCOBS wurde das *Java Application Building Center* (JavaABC) verwendet, welches ein Werkzeug zur grafischen Modellierung von Prozessen ist. Diese grafisch modellierten Prozesse werden in generierten Code überführt. JavaABC bietet eine besonders gute Anpassbarkeit an Veränderungen der zu Grunde liegenden Daten. Es stellte sich jedoch heraus, dass sich die Struktur der DBLP-Daten seit der Entwicklung von oCOBS nicht verändert hat.

StAX

Java EE 5 bietet verschiedene APIs für die Verarbeitung von XML-Dateien an. Dabei sind insbesondere Folgende zu unterscheiden:

- *DOM (Document Object Model)* - Die XML-Datei wird komplett in den Speicher geladen und in eine baumartige Datenstruktur überführt. Dies hat zwar einen größeren Speicherplatzbedarf zur Folge, ermöglicht aber einen sehr komfortablen Zugriff auf die Inhalte.
- *SAX (Simple API for XML)* - Im Gegensatz zur baumbasierten DOM-API erfolgt hier die Verarbeitung der XML-Datei ereignisgesteuert. Die Daten werden dabei nicht im Hauptspeicher abgelegt, sondern sequenziell durchlaufen. Für jedes XML-Element wird dann eine entsprechende Methode aufgerufen, die die weitere Verarbeitung übernimmt. Dadurch wird zwar die Geschwindigkeit deutlich gesteigert, allerdings mit dem Nachteil der unkomfortableren Benutzbarkeit.
- *StAX (Streaming API for XML)* - StAX versucht einen Mittelweg zwischen DOM- und SAX-API zu gehen. Wie bei SAX wird auch hier die XML-Datei sequenziell abgearbeitet, allerdings nicht nach dem *push*-, sondern nach dem *pull*-Prinzip. Der Programmcode steuert nun den Ablauf des Parsers, indem ein *Cursor* über das Dokument "geschoben" wird. Dabei kann jeweils das Element, an dem sich der Cursor momentan befindet, bearbeitet werden, bevor der Cursor zum Nächsten bewegt wird. Die Verarbeitungsgeschwindigkeit entspricht der von SAX bei einer einfacheren Benutzung (allerdings nicht so flexibel wie DOM).

Bei der Entscheidung, welche API für den oDOBS-Import Verwendung finden soll, musste die DOM-API sofort ausgeschlossen werden. Da die zu verarbeitende XML-Datei fast 700 MB groß ist (und stetig wächst) ist der Speicherplatzbedarf für eine Ablage im Hauptspeicher (inkl. des für die Datenstruktur notwendigen Overheads) zu hoch.

Die Entscheidung zwischen SAX- und StAX-API hatte die Projektgruppe, die den ursprünglichen Import implementiert hat, zugunsten von StAX getroffen. [8, Seite 51]

Javabib

Javabib [6] ist der wohl verbreitetste BibTeX-Parser und wurde von Johannes Henkel entwickelt. Er wird unter anderem von dem Tool BibTeX-To-RDF verwendet, mit dessen Hilfe BibTeX-Dateien in das RDF-Format exportiert werden können. Javabib bearbeitet einen Inputstream, aus dem es bekannte BibTeX-Konstrukte extrahiert. Diese können anschließend in ein eigenes Datenformat überführt werden. Durch die

an dieser Stelle sehr spärliche Dokumentation der Vorgängerprojektgruppen ist es nicht offensichtlich, warum sich diese für Javabib als BibTeX-Parser entschieden haben. Die Vermutung liegt allerdings nahe, dass dies aus Mangel an brauchbar dokumentierten Alternativen geschah.

7.3. Importvorgang Ausführen

Um bei oDOBS einen Importvorgang anzustoßen, gibt es zwei Möglichkeiten. Zum einen gibt es den im Wiki der Vorgängerprojektgruppe [3] beschriebenen Weg per Konsole. Genaueres dazu findet sich in Kapitel 7.3.2 auf der nächsten Seite. Darüber hinaus existiert ein Import-Servlet (siehe Kapitel 7.3.1), welches auf einer lokalen oDOBS-Instanz unter `http://localhost:8080/oDOBS_import/` zu erreichen ist und sowohl den DBLP-Publikationsimport, als auch den DBLP-Kategorieimport und den BibTeX-Import anstoßen kann.

7.3.1. Import-Servlet

Zusätzlich zur dokumentierten Variante des konsolenbasierten Imports existiert ein Import-Servlet, welches von einer der Vorgängerprojektgruppen erstellt wurde, allerdings weder in den Admin-Client eingebunden wurde, noch in der Dokumentation zu oDOBS auftauchte. Dabei eignet sich das Import-Servlet eigentlich für sämtliche unterstützten Importvorgänge. Der DBLP-Publikationsimport bietet sowohl die Möglichkeit, die `DBLP.xml` direkt vom Server der DBLP herunterzuladen und den Importvorgang zu starten, als auch die Möglichkeit, eine lokal verfügbare XML-Datei, welche in DBLP-Syntax vorliegen muss, zu importieren. Letzteres ist zum Beispiel geeignet, um Testdaten zu importieren, ohne den kompletten DBLP-Datenimport auszuführen. Stattdessen wird dann z.B. eine gekürzte Datei als Quelle angegeben.

Auch der DBLP-Kategorieimport bietet die Möglichkeiten, die benötigten Daten direkt vom DBLP-Server herunterzuladen oder eine lokale Variante der `DBLP_bht.xml` zu verwenden. Auch hier besteht die Möglichkeit, auf gekürzte oder anderweitig angepasste Daten zurückzugreifen.

Die dritte unterstützte Variante ist der BibTeX-Import. Hier ist die Grundlage eine `.bib`-Datei, die sich entweder lokal auf der Festplatte befinden oder online verfügbar sein kann. Hierdurch kann man sowohl eigene lokale Literaturverzeichnisse einspielen, als auch solche, die sich im Onlineangebot diverser Verlage finden. Als Ergänzung würde sich hier der Import einer textuellen Eingabe im BibTeX-Format anbieten, um kleine Datenmengen, ohne den Umweg über eine Datei, schnell importieren zu können.

Problematisch ist die schlechte Rückmeldung des Import-Servlets an den Benutzer. Es wird lediglich mitgeteilt, dass der Importvorgang erfolgreich initiiert wurde

und der Benutzer wird auf die Server-Logs für weitere Informationen verwiesen. Allerdings erfolgt diese Nachricht auch, wenn der Importvorgang fehlerhaft ist, zum Beispiel wenn eine nicht existierende Datei für den BibTeX-Import angegeben wird oder der Datenbankserver nicht gestartet ist.

7.3.2. Konsolenbasierter Import

Soll auf einer neu angelegten oCOBS-Instanz ein Import ausgeführt werden, so sind vorher einige Vorkehrungen zu treffen. Es werden diverse Bibliotheken für den Importvorgang benötigt. Diese sollten heruntergeladen und im entsprechenden lib-Verzeichnis abgelegt werden. Später müssen diese beim Aufruf des Imports im classpath-Attribut der .jar-Datei angegeben werden. Weiterhin muss eine oCOBS-Benutzer namens *importer* existieren, der mit Administrator-Rechten ausgestattet ist. In einer Konfigurationsdatei ist das korrekte Datenbanksystem inklusive der zugehörigen Zugangsdaten anzugeben. Sind diese Vorkehrungen getroffen, kann der Import über einen Konsolenbefehl gestartet werden. Detailliertere Informationen hierzu finden sich unter [13].

7.3.3. Performance & Stabilität

Nach dem Start eines Importvorgangs bei oCOBS werden die ersten Datensätze noch mit einer zufriedenstellend hohen Geschwindigkeit eingelesen und in die Datenbank übernommen. Allerdings nimmt die benötigte Zeit für den Import eines Datensatzes kontinuierlich zu. Nach etwa 30 Stunden bricht der Importvorgang vorzeitig ab. Das Hochrechnen der bis dahin vergangenen Zeit auf die Gesamtzahl der Datensätze ergibt eine Laufzeit von mehreren Wochen für einen vollständigen Import. Währenddessen entstehen mehrere Gigabyte große Logdateien. Eine Erhöhung des dem Importvorgang zugeteilten Arbeitsspeichers konnte die Performance zwar leicht steigern, führte aber auch nicht zu zufriedenstellenden Ergebnissen.

Als Hauptursache für die schlechte Performance des Imports werden die umfangreiche Loggingfunktion und die Doublettensuche vermutet. Die Loggingfunktion arbeitet sehr feingranular und erzeugt dadurch mehrere Gigabyte große Logdateien, welche extrem unübersichtlich sind und durch ihren Umfang den Importvorgang deutlich ausbremsen. Zusätzlich wird die Doublettensuche aufgerufen, welche den gesamten Datensatz auf eine mögliche Doublette der gerade zu importierenden Publikation durchsucht. Dies ist bei besonders großen Datensätzen höchst problematisch, da somit die Laufzeit mit jeder zusätzlichen Publikation stark ansteigt. Weiterhin werden unnötige Datenbank- und Webanfragen getätigt. Wenn zum Beispiel mehrere Autoren den gleichen Namen besitzen, wird von der DBLP ein Suffix vergeben, um diese zu identifizieren. Dieses Suffix wird jedoch bei oCOBS nicht gesondert gespeichert. Auch das bei der Suchfunktion aufgetretene Speicherleck (siehe Kapitel 5 auf Seite 33) könnte hier zu Problemen führen.

Schlussendlich stellt sich die Frage, warum oCOBS lediglich einen Komplettimport der DBLP-Daten unterstützt. Für einen Teilimport müsste die ca. 700 MB große `DBLP.xml` von Hand gekürzt werden.

7.4. Fazit

Nach anfänglichen Einarbeitungsschwierigkeiten war es uns schließlich möglich, den Datenimport zu starten. Ein vollständiger Durchlauf gelang uns allerdings nicht. Dies ist vermutlich darauf zurückzuführen, dass sich die zu importierende Datenmenge seit dem letzten Import nahezu verdoppelt hat. Dadurch treten die in Abschnitt 7.3.3 beschriebenen Probleme besonders in den Vordergrund.

Die Möglichkeit eines Teilimports besteht nicht. Zwar überprüft der Import, ob einzulesende Daten bereits in der Datenbank vorhanden sind, die Art der Implementierung bewirkt jedoch kaum eine Leistungssteigerung. Dies führt dazu, dass auf den initialen Import folgende Importvorgänge (um den Datenbestand aktuell zu halten) praktisch die gleiche, hohe Laufzeit benötigen. Das ist vermutlich auch der Grund dafür, dass in der Vergangenheit keine laufenden Aktualisierungen durchgeführt wurden.

Teil III.

Weiterentwicklungen der PG 540

8. Benutzerfunktionen

Es gibt eine Reihe von Möglichkeiten, um die Inhalte von oCOBS innerhalb eines gewissen Rahmens dynamisch anpassen zu können. Im Falle einer Webseite, die sich der Literaturrecherche widmet, bietet sich eine Vielzahl möglicher Funktionen an, die den anderen Benutzern die Recherche erleichtern kann.

Zu diesen Funktionen gehören beispielsweise die Bewertung und die Kommentierung von vorhandenen Publikationen. Durch diese können angemeldete Benutzer anderen Benutzern Informationen über die Qualität, Stärken, Schwächen und inhaltliche Schwerpunkte der jeweiligen Publikationen mitteilen. Dies trägt zur Qualitätssteigerung der oCOBS Plattform bei und kann das Angebot für alle Benutzer interessanter und übersichtlicher gestalten. Jedoch sollte darauf geachtet werden, dass die Interaktivität der Seite nicht missbraucht werden kann. Dies erfordert eine sichere Implementierung der Funktionen und eine umfassende Betreuung durch die Administratoren der Webseite.

Die folgenden Kapitel befassen sich mit der Umsetzung dieser Benutzerfunktionen und beschreiben Lösungsstrategien und verwendete Technologien.

8.1. Bewertung

Es ist angemeldeten Benutzern vorbehalten, Publikationen zu bewerten. Durch diese Bewertungen kann ein Benutzer schnell und einfach eine Publikation beurteilen und sich über die durchschnittliche Beurteilung anderer Benutzer informieren.

Wie in Abschnitt 1.4 auf Seite 7 erläutert, geschieht die Bewertung durch die Vergabe von bis zu fünf Sternen, dabei ist ein Stern die niedrigste Bewertung und fünf die höchste. Sowohl die eigene Bewertung des aktuellen Benutzers als auch die durchschnittliche Bewertung werden in Form von Sternen auf jeder Publikationsseite angezeigt. Zusätzlich wird auch die durchschnittliche Bewertung als Dezimalzahl neben den Sternen angezeigt. Ein Benutzer kann jede Publikation nur einmal bewerten. Allerdings ist es ihm jederzeit möglich, seine Bewertung zu ändern.

Da die Realisierung von Bewertungen nicht aufwendig ist, schien die Implementierung einer eigenen Lösung angemessen. Es wurde entschieden, dass für die Bewertung eine Klasse angelegt wurde, welche den Verweise auf die bewertete Publikation und den bewertenden Benutzer enthält und natürlich die Bewertung an sich. Um die Performance zu erhöhen, werden die durchschnittliche Bewertung und die gesamte Anzahl der Bewertungen einer Publikation zusätzlich in dieser Publikation

gespeichert. Durch diese Optimierung ist es nicht notwendig, bei jedem Aufruf einer Publikationsseite sämtliche Bewertungen zu durchlaufen und einen Mittelwert der Bewertungen und die Anzahl der Bewertungen zu bestimmen. Stattdessen ist es möglich, dies zusammen mit den anderen Publikationsdaten auszulesen. Des Weiteren lässt sich die durchschnittliche Bewertung auch bei der Änderung der Bewertung schnell berechnen.

Die Bewertung wurde mit Hilfe von *JavaScript* und *AJAX* realisiert. JavaScript ist eine clientseitige Skriptsprache und kann im Webbrowser des Anwenders ausgeführt werden. Da der Webbrowser keine Anfrage an den Server schicken und die Antwort nicht abwarten muss, werden einerseits viele Vorgänge beschleunigt und anderseits die Belastung der Kommunikationswege und des Servers reduziert (vgl. [17, S. 45]). AJAX wurde entwickelt, um die Performance der Webapplikationen auch zu verbessern. Statt einer vollständigen Webseite werden nur die nötigen neuen Informationen vom Server gesendet und anschließend in die bereits geladene Webseite eingebaut (vgl. [17, S. 47]). Bei der Implementierung wurden die JavaScript-Funktionen und die AJAX-Interaktionen durch das umfangreiche Framework *jQuery* [60] erleichtert, wobei *JSON* [30] als Datenaustauschformat verwendet wurde.

Eine Bewertung wird durch eine AJAX-Interaktion an den Server gesendet. Auf der Serverseite wird diese neue Bewertung entsprechend behandelt. Schließlich wird die neu berechnete durchschnittliche Bewertung der Publikation zurück an den Client gesendet und dann in die bestehende Publikationsseite eingebaut.

Web-Oberfläche der Bewertung

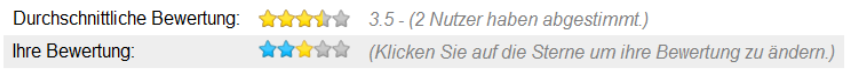

Abbildung 8.1.: Bewertung der Publikation

Abbildung 8.1 zeigt den Bereich der Publikationsseite, der für die Bewertung angelegt wurde. Die erste Zeile zeigt die durchschnittliche Bewertung sowohl in Form von Sternen als auch als Dezimalzahl. Die Sterne werden bis zur durchschnittlichen Bewertung gelb statt Grau dargestellt. Hier steht ein Stern für eine Stufe. Zusätzlich wird auch die Anzahl der Benutzer angezeigt, die schon eine Bewertung abgegeben haben.

Nur ein angemeldeter Benutzer kann die zweite Zeile sehen, um seine Bewertung abzugeben oder zu ändern. Falls der Benutzer noch nicht abgestimmt hat, werden alle fünf Sterne grau dargestellt. Falls er schon eine Bewertung abgegeben hat, werden die Sterne bis zu seiner Bewertung gelb dargestellt. Die blauen Sterne zeigen die Bewertung, die der Benutzer abgeben oder ändern möchte.

8.2. Kommentare

Das Kommentieren von Publikationen ist eine weitere Benutzerfunktion, die in oCOBS integriert werden sollte. Jeder Benutzer kann in seinem Kommentar seine Meinung zu einer Publikation in Textform äußern oder auf den Kommentar eines anderen Nutzers eingehen.

Die Anforderungen an die Kommentarfunktion wurden bereits in Abschnitt 1.4 auf Seite 7 erwähnt. Die abgegebenen Kommentare werden auf der zugehörigen Publikationsseite angezeigt. Die aktive Verwendung der Kommentarfunktion ist angemeldeten Benutzern vorbehalten. Weiterhin ist es einem angemeldeten Benutzer möglich, jederzeit seine alten Kommentare zu löschen, unpassende oder beleidigende Kommentare an den Administrator zu melden und mit einem Klick das öffentliche Profil des Verfassers eines Kommentars einsehen zu können. Nicht angemeldete Benutzer haben lediglich die Möglichkeit, die Kommentare anderer Benutzer einzusehen.

Ähnlich wie bei der Bewertungsfunktion hat die PG sich auch hier für die Implementierung einer eigenen Lösung entschieden, dafür wurde eine Klasse angelegt, welche den Verweise auf die kommentierte Publikation und den kommentierenden Benutzer enthält. Ein Kommentar, der nicht leer ist, wird zusammen mit der ID der Publikation durch eine AJAX-Interaktion an den Server gesendet und dann in die Datenbank gespeichert. Bei dem Löschen wird die ID des Kommentars an den Server gesendet. Der Kommentar wird auf der Serverseite gelöscht und das entsprechende DOM-Element auf der Clientseite wird ebenfalls entfernt. So kann die ganze Publikationsseite nicht neu geladen werden. Des Weiteren muss noch die in oCOBS bestehende Feedback-Funktion erweitert werden, um das Melden eines Kommentars zu realisieren.

Web-Oberfläche des Kommentars

Abbildung 8.2 auf der nächsten Seite illustriert die Web-Oberfläche des Kommentars für einen angemeldeten Benutzer. Der Benutzer gibt seinen Kommentar in eine Textbox ein, um zu kommentieren. Unter den Textboxen werden die Kommentare chronologisch absteigend angezeigt. Für jeden Kommentar werden der vom Verfasser gewählte Anzeigename, das Datum und der Inhalt angezeigt. Ein Klick auf den angezeigten Namen öffnet die Profilseite des Verfassers. Weiterhin gehört genau ein Symbol zu jedem Kommentar, welches entweder für das Löschen eigener Kommentare oder für das Melden fremder Kommentare steht.

Wenn ein Benutzer sich nicht angemeldet hat, werden die Textbox für die Abgabe eines Kommentars und die Symbole für Löschen und Melden ausgeblendet. Zudem wird der Name des Verfassers nur als reiner Text angezeigt und nicht zur Profilseite verlinkt.

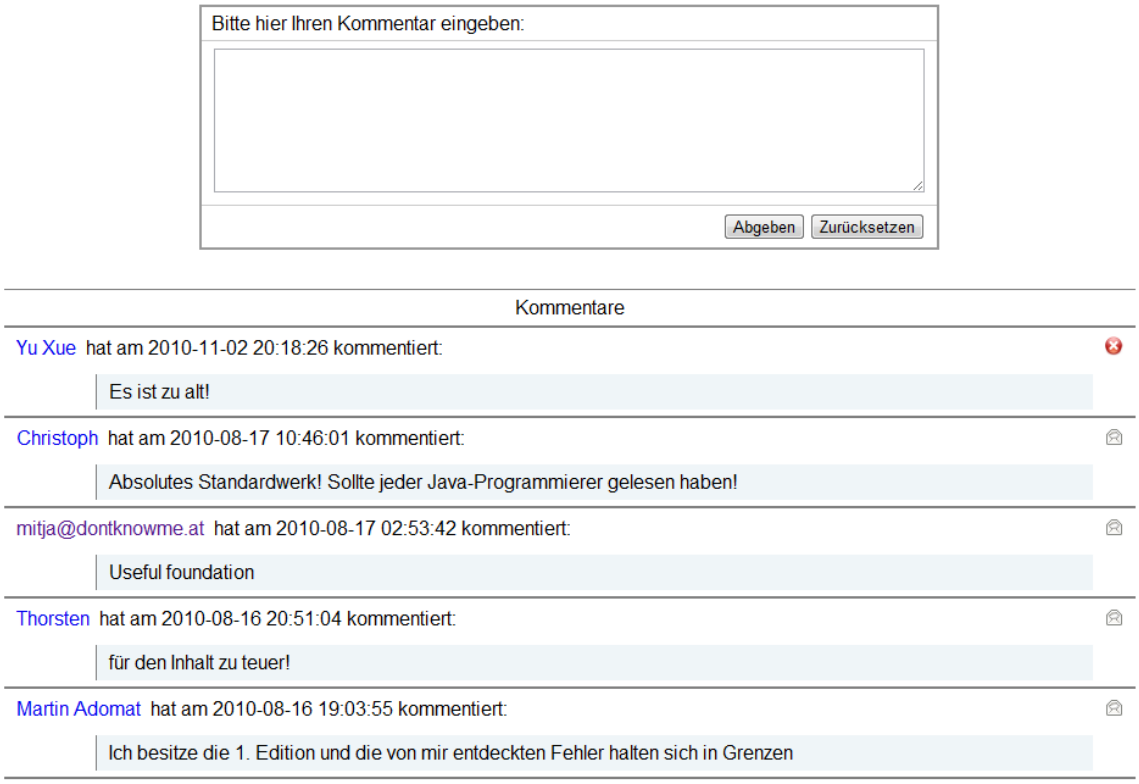

Abbildung 8.2.: Kommentar der Publikation

8.3. Tagging

Als Tagging wird das Anhängen von *Metadaten* jeglicher Art an eine Information oder „einen Informationsträger“, bezeichnet. Unter Metadaten versteht man zusätzliche Informationen, welche die gegebenen Daten bzw. Informationen selbst beschreiben. Als Beispiel könnte man sich hier einen Zeitungsartikel (Datum) über politische Debatten im Internet vorstellen, der noch zusätzlich thematisch passende Informationen (Metadaten) enthält wie beispielsweise „Angela Merkel“, „Bundeskanzlerin“, „CDU“ usw. Der Zeitungsartikel lässt sich danach mit Hilfe der Metadaten bei einer Suche einfacher finden, da er nun auch über diese zusätzlichen Informationen gefunden werden kann.

Das Prinzip des Taggings findet sich meist auf Internetseiten mit einem sozialen Netzwerk. Die Nutzer dieses Netzwerks sind in der Regel diejenigen, die den Informationsträgern die neuen Tags zuweisen. Das hat den Vorteil, dass die Last der Tag-Erstellung auf viele verschiedene Personen aufgeteilt wird und sich die Menge der Tags dynamisch weiterentwickelt. Außerdem werden dadurch bessere Suchergebnisse erzielt, da die Tags von denen vergeben werden, die auch mit dem Informationsträger arbeiten. Diese freie, dynamische Weiterentwicklung durch den Nutzer

hat allerdings auch einen entscheidenden Nachteil, die Kategorisierung der Wörter wird stark erschwert. Zum einen kann ein Wort in mehreren Formen gespeichert sein (Buch, Bücher, books ...), und zum anderen ist die Bedeutung einer Metainformation vielleicht nur im Kontext klar. Während also „apple“ im Deutschen für Apfel steht, erwartet man in der Informatik eher die Firma *Apple Inc.*.

Mit den derart gesammelten Tags ist es nun möglich eine *Tag-Cloud* zu erzeugen, wie z.B. bei Flickr [41]. In einer Tag-Cloud werden alle angehängten Metadaten in unterschiedlicher Wortgröße dargestellt, wobei die Größe der Schlagwörter/Tags visuell ihre Relevanz untermauert. Als relevant wird in den meisten Fällen die Häufigkeit des Tags angesehen. Wenn also beispielsweise 100 Artikel mit dem Schlagwort „Psychologie“ und 20 Artikel mit dem Schlagwort „Training“ verknüpft wurden, dann bekommt „Psychologie“ eine deutlich größere Schrift in der Tag-Cloud als „Training“.

Eine bekannte Seite, auf der Tags genutzt werden, ist *„delicious.com“* [40]. Hier können Nutzer ihre favorisierten Internetseiten anderen Nutzern zur Verfügung stellen und sich die Favoriten anderer Nutzer ansehen. Diese Links können dann durch die Nutzergemeinde von delicious.com mit (weiteren) Tags versehen werden. Über diese Tags lassen sich dann weitere Links finden, denen dasselbe Tag gegeben wurde. Außerdem gibt es bei Tags auch untereinander eine sogenannte „Verwandschaft“. Verwandt sind solche, die häufig paarweise auftauchen. Dies unterstreicht die inhaltliche Zusammengehörigkeit zweier Tags und sorgt dafür, dass das Augenmerk auch auf ähnliche Tags fällt.

Die Aufgabe der PG540 ist es, oCOBS um diese Eigenschaft eines sozialen Netzwerkes zu erweitern und den Dienst somit für alle Nutzer attraktiver und interaktiver zu gestalten. Dazu muss sich präzise über die Realisierungsmöglichkeiten informiert werden, um daraus die für oCOBS geeignetste Variante auszuwählen.

Zur Realisierung von Tags in einem J2EE Projekt kann man zum einen auf vorgefertigte Frameworks zurückgreifen oder zum anderen Tagging selbst von Hand umsetzen. In den folgenden Abschnitten werden diese Lösungsmöglichkeiten genauer erläutert, sowie deren Vor- und Nachteile abgewogen.

8.3.1. Eigenimplementierung

Als Nächstes werden einige Aspekte einer eigenen Implementierung von Tagging betrachtet. Der womöglich größte Vorteil bei einer Eigenimplementierung ist die Flexibilität, die den Programmierern bei der Realisierung bleibt. Dadurch kann die Funktionalität konkret an das eigene Projekt angepasst werden und nicht umgekehrt. Allerdings ist zu beachten, dass dies keine „best practice“ Lösung ist.

Bei der Auswahl von Tags kann es häufig zu Redundanzen kommen. Um diesen Wiederholungen vorzubeugen, gibt es die Möglichkeit der „halb automatischen Tagvergabe“. Das bedeutet, dass zum Zeitpunkt der Vergabe von Tags durch den Nutzer bereits Vorschläge aus der Datenbank, in Form von vorhandenen Tags, ge-

macht werden. Dies würde allerdings wieder ein Problem aufwerfen, und zwar muss erst einmal ein initialer Datenbestand von Tags existieren, damit hieraus Tags vorgeschlagen werden können. Es bietet sich hierfür an, entweder die Tags aus anderen Bibliotheksverwaltungen (wie z.B. CiteULike) zu holen, was allerdings zu einer gewissen Abhängigkeit von weiteren externen Internetseiten führt, oder aber die Titel der einzelnen Artikel nach initialen Tags zu durchsuchen.

Da es sich um „social“ tagging handelt, müssen die Nutzer natürlich auch neue Tags vergeben können. Der Nutzer ist also nicht darauf beschränkt, den Vorschlag aus der eben genannten Vorschlagsliste anzunehmen, sondern kann diese dementsprechend erweitern. Außerdem ist es unumgänglich, dass ein Nutzer die Möglichkeit bekommt, vorhandene Tags anderer Nutzer zu melden, falls diese auf irgendeine Art und Weise unpassend vergeben wurden.

Des Weiteren können auch Verwandtschaften von Publikationen leicht erkannt werden. Eine Verwandtschaft zwischen zwei Publikationen besteht dann, wenn diese dieselben Tags zugewiesen bekommen haben. Ein ebenfalls oft verwendetes Feature ist die Speicherung einer Ansammlung von favorisierten Tags und Artikeln, über die der Nutzer dann möglichst leicht seine wichtigsten Artikel erreicht.

Konkret bedeutet eine eigene Realisierung von Tagging, dass einige Datenbankfelder hinzugefügt werden müssen, damit die zusätzlichen Informationen der Publikation zugewiesen werden können. Allerdings muss die Funktionalität des Taggings vollständig selbst implementiert werden (Tags hinzufügen & löschen, Tag-Clouds erzeugen & anzeigen ...), was ansonsten von einem Framework unterstützt werden kann. Für diese Realisierung sind dennoch keine großen Umstrukturierungen in der Datenbank notwendig und die Einarbeitung in ein neues Framework und dessen Dokumentation entfällt ebenfalls. Somit ist auch der Neueinstieg in eine selbst entwickelte Tagging Realisierung einfacher.

8.3.2. Frameworks

Es existiert eine Vielzahl von vorgefertigten Frameworks, die zur Realisierung von Tagging in Webprojekten genutzt werden. Um eine Entscheidung über die Verwendung eines Frameworks zu treffen, sollten alle Vor- und Nachteile gegeneinander abgewogen werden. Außerdem sollte der Entwickler sich der Anforderungen, des Ist-Zustandes und der Ziele des eigenen Projekts bewusst sein.

Alle infrage kommenden Frameworks müssen also einer gründlichen Analyse unterzogen werden. Ansatzpunkte hierfür sind die Funktionalität, die ein Framework bietet, sowie die Verbreitung und Akzeptanz im Internet. Zusätzlich stellt sich die Frage nach möglichen Lizenzkosten, Verfügbarkeit des Quellcodes (Open Source?), der Qualität der Dokumentation, Erweiterbarkeit, bekannten Bugs und der benötigten Einarbeitungszeit.

Nach genauem Studium verschiedener Frameworks stellte sich heraus, dass für das Projekt oCOBS nur ein *RDF*-basiertes Framework infrage kommt. *RDF* sowie

zwei Beispielframeworks werden im folgenden Abschnitt erklärt.

8.3.2.1. RDF

Das *Resource Description Framework* (RDF) ist ein durch das W3C (World Wide Web Consortium) [44] standardisiertes Framework zur Beschreibung von Informationen über Ressourcen im Web. Es können sowohl die Metadaten als auch allgemeine Informationen dargestellt werden. Diese durch RDF beschriebenen Informationen können von Anwendungen eingelesen und verarbeitet werden. Dies bedeutet, dass ein Austausch von Daten zwischen verschiedenen Anwendungen ebenfalls verlustfrei realisierbar ist.

Das RDF speichert Informationen in Form von Aussagen. Eine RDF-Aussage ist ein Tripel, welches genau aus einem Subjekt, einem Prädikat und einem Objekt besteht. Das Subjekt ist die zu beschreibende Ressource, das Prädikat ist eine Eigenschaft des Subjekts und das Objekt ist der Wert dieser Eigenschaft. Beispielsweise beschreibt das Tripel („B“ „Vater“ „A“), dass A der Vater von B ist. Das Prädikat und das Objekt im Tripel sind auch Ressourcen. Als Identifikator für die Ressourcen verwendet RDF wegen der Eindeutigkeit *URIref*, welches aus „URI“ und „fragment identifier“ besteht. RDF Aussagen können auf verschiedene Art und Weise gespeichert werden, z.B. *N3* (menschenlesbares Format), *gerichteter Graph* (grafisches Format), *RDF/XML* (maschinenlesbares Format). Jede Ressource ist die Instanz von mindestens einer Klasse. Die Mengen der vordefinierten Eigenschaften (Prädikate) und Klassen werden Vokabeln genannt, die durch ein RDF Schema definiert werden können. Mit Hilfe von vorhandenen Aussagen können einige RDF-Softwarelösungen wie z.B. *Sesame* [39] neue Informationen erwerben. Eine ausführliche Vorstellung für den gerichteten Graph, RDF/XML, RDF Schema und einige Anwendungen befindet sich unter [20]. Dort gibt es auch viele Beispiele, die zum Verständnis des RDF Konzeptes beitragen können.

Mit Hilfe von RDF ist der Informationsgewinn leicht realisierbar. Auch Doubletten sind aufgrund der besseren Beschreibungen der Ressourcen einfach erkennbar. Ein weiterer Vorteil ist, dass die Anbindung des RDF-Frameworks an relationale Datenbanken möglich ist. Außerdem gibt es eine Vielzahl verschiedener Open-Source Frameworks (z.B. *Jena* [27] und *Sesame* [39]) für RDF-Anwendungen. Ein Nachteil von RDF ist allerdings, dass dieses Framework am besten bei neuen Projekten eingesetzt wird, die noch kein Datenbankmodell besitzen, da ein für RDF geeignetes Datenbankmodell komplett anders aufgebaut ist. Wie im folgenden Abschnitt gezeigt wird, reicht ein Hinzufügen von neuen Tabellen bzw. die Änderung der bestehenden Tabellen nicht aus. Deswegen ist ein komplettes Redesign des Datenbankmodells notwendig. Dadurch würden auch exorbitant viele Änderungen in schon bestehendem Quellcode anfallen.

Die beiden Frameworks Jena und Sesame unterstützen eine Vielzahl von relationalen Datenbanken (z.B. MySQL, PostgreSQL). Das Datenbankmodel von Jena

besteht aus zwei Aussagetabellen (`Jena_GiTj_Stmt`, `Jena_GiTj_Reif`) und sechs Systemtabellen (`Jena_Sys_Stmt`, `Jena_Long_Lit`, `Jena_Long_URI`, `Jena_Prefix`, `Jena_Graph`, `Jena_Mutex`). Im Gegensatz dazu versucht Sesame für jede Klasse und Eigenschaft eine Tabelle zu erzeugen, falls die Anzahl der Tabellen die vorgegebene Grenze noch nicht erreicht. Das heißt, dass die Anzahl der Tabellen von der Anzahl der Klassen und Eigenschaften abhängt. Dokumentation und Beispielcode sind auf der jeweiligen Webseite der Frameworks erreichbar.

RDF ist eigentlich nicht für Tagging konzipiert worden. Es ist allerdings möglich, die zu taggenden Ressourcen, welche bei oDOBS zum Beispiel Publikationen sein können, und die zu vergebenen Tags mit Hilfe von RDF-Aussagen zu beschreiben. Die RDF-Aussagen für Tagging können sowohl in XML-Dateien als auch in einer relationalen Datenbank gespeichert werden.

8.3.3. Realisierung

Die Projektgruppe hat sich für eine eigene Realisierung des Taggings entschlossen, um die Flexibilität zu erhalten und somit eine weitere komplexe API und eventuell größere Änderungen am Datenmodell zu vermeiden. Ein RDF-Framework findet somit keine Verwendung.

Änderungen am Datenmodell

Um Tagging zu realisieren, wurde eine Änderung am Datenmodell notwendig. Eine Person taggt eine Publikation mit einem Schlagwort. Es wird also der taggende Benutzer, die getaggte Publikation und das vergebene Schlagwort benötigt um einen Tagvorgang zu persistieren. Alle drei Informationen werden in einer neuen Klasse namens `Tag` gespeichert. Um eine direktere Zugriffsmöglichkeit zu bieten, wurde die Klasse `Publikation` um eine Liste aller ihr zugeordneten Tagobjekte erweitert. Ähnlich wurde auch der Klasse `OdobsUser` eine Liste aller von diesem Benutzer vergebenen Tag-Objekte hinzugefügt.

Tag-Clouds

Der Beweggrund Tags zu vergeben ist häufig die Organisierung des eigenen Datenbestandes. Der Benutzer möchte gerne Publikationen, für die er sich interessiert, bezüglich ihres Inhaltes ordnen. Es ist daher wichtig, dem Benutzer den Zugriff auf eine geordnete Liste der von ihm getaggten Publikationen zu bieten. Dies wird über eine Tag-Cloud auf der Profilseite des Benutzers realisiert.

Durch die Tags sollen für andere Benutzer sichtbare Informationen an Publikationen gehängt werden. Diese sollen auf der Publikationsseite derart dargestellt werden, dass die Relevanz der Tags herausgehoben wird. Auch dies wird mittels einer Tag-Cloud realisiert.

Außerdem soll durch die vergebenen Tags eine Art „Taglandschaft“ erzeugt werden, welche die Navigation durch die enthaltenen Publikationen erleichtert. Hierfür wird auf der Startseite eine globale Tag-Cloud erzeugt.

Benutzer-Tag-Cloud Die Benutzer-Tag-Cloud erscheint auf der Profilseite eines Benutzers und gibt die Gesamtheit der von ihm getaggten Publikationen wieder. Sie ist für alle Benutzer offen einsehbar und ermöglicht dadurch, dass Benutzer von Tagvorgängen anderer Benutzer profitieren. Abbildung 8.3 zeigt das Profil eines anderen Benutzers inklusive dessen Benutzer-Tag-Cloud.

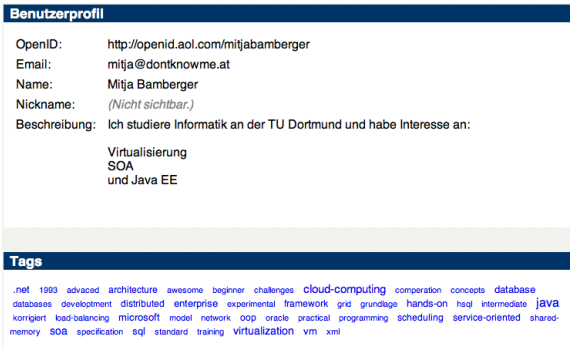

Abbildung 8.3.: Beispiel: Benutzer-Tag-Cloud

Die unterschiedliche Gewichtung von häufig oder seltener durch den Benutzer vergebenen Tags wird durch unterschiedlich große Schriftgrößen realisiert. Aufgrund einer vermutlich überschaubaren Datenmenge hat sich die Projektgruppe entschieden, alle vergebenen Tags in die Tag-Cloud aufzunehmen und jedes Mal, wenn ein Tag vergeben wurde, die Schriftgröße des Schlagwortes um eins zu erhöhen. Damit die Schlagworte nicht zu klein oder zu groß ausfallen, gibt es eine obere und eine untere Schranke.

Ein Klick auf einen Tag zeigt alle Publikationen, die von diesem Benutzer hiermit getaggt wurden. Um dies erreichen zu können, muss die Suchfunktion dahingehend erweitert werden, dass es möglich ist nach Tags von einem bestimmten Benutzer zu suchen.

Publikations-Tag-Cloud Die Publikations-Tag-Cloud beinhaltet sämtliche einer Publikation zugeordneten Tags. Diese ermöglicht es Benutzern, die Themen bzw. Inhalte einer Publikation überblicken zu können.

Auch hier gehen wir von einer überschaubaren Menge an vergebenen Tags aus, weshalb alle vergebenen Tags angezeigt werden. Für die Gewichtung der vergebenen Tags wird die gleiche Methode wie bei der Benutzer-Tag-Cloud verwendet. Somit ist ersichtlich, welche Tags für eine Vielzahl von Benutzern relevant sind, und welche nur von wenigen Benutzern gewählt wurden.

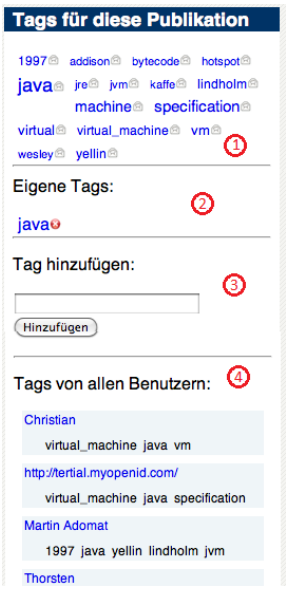

Abbildung 8.4.: Beispiel: Tag-Cloud einer Publikation

Abbildung 8.4 zeigt die zugehörige Web-Oberfläche. An der mit `1` markierten Stelle befindet sich die oben angesprochene Tag-Cloud. Zusätzlich befindet sich an jedem Element ein Icon, welches es ermöglicht, unerwünschte Tags zu melden. Dies kann simple Gründe wie Schreibfehler haben, aber auch auf gezielte Fehlinformation, oder gar unerwünschte Werbung oder gesetzeswidrige Inhalte zurückzuführen sein. Will ein Benutzer einen Tag melden, so öffnet sich eine neue Seite, auf der er den Grund seiner Beschwerde angeben soll und auch seine E-Mail-Adresse hinterlegen kann. Für diese Beschwerde wird daraufhin ein Task für den Administrator erstellt, welcher von diesem zu überprüfen ist.

Um zu verhindern, dass Leute versuchen einen Tag mehrfach zu vergeben und ihm einen Überblick über seine bereits getätigten Tags zu bieten, gibt es eine Liste von diesen. In Abbildung 8.4 befindet sich diese Liste an der mit `2` markierten Stelle. An den Tags in dieser Liste befindet sich jeweils ein Icon um eigene Tags zu löschen. Dies ist vor allen Dingen nützlich, wenn ein Schreibfehler zu spät bemerkt wird und vermeidet es einen Administrator hinzuziehen zu müssen.

An der mit `3` markierten Stelle kann der Benutzer neue Tags hinzufügen. Um verschiedene Schreibweisen zu unterbinden, werden sämtliche Buchstaben der Eingaben in Kleinbuchstaben umgewandelt. Außerdem wird bei jeder Eingabe überprüft, ob der Benutzer dieses Schlagwort bereits zu dieser Publikation vergeben hat. Ist dies der Fall, so wird es nicht erneut hinzugefügt. Dies verhindert die doppelte Vergabe von Tags. Die eigentliche Tag-Cloud (an der Stelle `1`) dient als Vorschlagsliste für das Hinzufügen weiterer Tags. Aus Gründen der Nachvollziehbarkeit ist es nur angemeldeten und eingeloggten Benutzern erlaubt Tags zu vergeben.

Um es Benutzern zu erleichtern andere Benutzer mit ähnlichen Interessen zu finden hat sich die Projektgruppe entschlossen eine Taghistorie für jede Publikation zu erstellen. In Abbildung 8.4 auf der vorherigen Seite befindet sich die Taghistorie an der mit 4 markierten Stelle.

Globale Tag-Cloud Auf der Startseite befindet sich die globale Tag-Cloud (Siehe Abbildung 8.5, welche die gesamte Tag-Landschaft des Systems abbildet. Dies dient zur gezielten Navigation durch die getaggten Publikationen und erleichtert es ähnliche Publikationen zu finden.

Abbildung 8.5.: Beispiel: Globale Tag-Cloud

Anders als bei den beiden bisherigen Tag-Clouds wird bei der globalen Tag-Cloud eine weitaus größere Menge an enthaltenen Tags erwartet, da sämtliche vergebene Tags erfasst werden. Dies hat drei wichtige Änderungen zur Folge.

Zunächst muss die Strategie zur Visualisierung von wichtigeren und unwichtigeren Tags geändert werden, da sonst viele Tags sehr schnell die maximale Gewichtung erhalten würden. Daher hat sich die Projektgruppe entschlossen bei der globalen Tag-Cloud auf eine prozentuale Gewichtung zu setzen. Die Schriftgröße eines Tags ist davon abhängig, wie hoch sein Auftreten im Vergleich zu dem Tag mit dem maximalen Auftreten ist. Hierdurch wird auch bei hohem Datenbestand eine gute Visualisierung der Relevanz von Tags erreicht.

Der zweite Punkt ist, dass es bei den zu erwartenden Datenmengen und der Anzahl der zu erwartenden verschiedenen Tags unabdingbar ist, die Anzahl der hier angezeigten Tags zu begrenzen. Daher zeigt die globale Tag-Cloud nur eine beschränkte Anzahl der am häufigsten vergebenen Tags an.

Die letzte notwendige Änderung ist die Wichtigste. Hinter den beiden anderen Tag-Clouds steckt eine nicht persistierte Datenstruktur, welche bei Seitenaufruf mit Inhalt befüllt wird. Bei den anderen Tag-Clouds ist dies durch den eher geringen Datenumfang durchaus praktikabel. Da die globale Tag-Cloud allerdings sämtliche im System vorhandenen Tags enthält würde ein Auslesen der Taglisten jeder Publikation zu großen Performanceproblemen führen. Um dieses Problem zu bewältigen, hat die Projektgruppe als zusätzliche Datenstruktur eine Liste von allen Schlagwörtern

und die Anzahl deren Auftretens im System, angelegt. Diese wird parallel zu den anderen Daten geführt und auch persistiert. Daher ist es notwendig diese zu pflegen und sie bei jeder Änderung im Datenbestand zu aktualisieren. Konkret muss das jeweilige Auftreten der Schlagwörter bei der Erstellung eines entsprechenden Tags erhöht und beim Löschen des jeweiligen Tags reduziert werden.

Initiale Tagvergabe

Um für neu hinzugefügte, oder bislang nicht von Benutzern getaggte, Publikationen auch Tagvorschläge anbieten zu können, hat sich die Projektgruppe dazu entschlossen mit dem DBLP-Publikationsimport (vgl. Kapitel 7 auf Seite 40) eine initiale Tagvergabe durchzuführen. Um dies zu realisieren, wird der Publikationstitel auf mögliche Schlagworte durchsucht. Hierfür wird der Publikationstitel in alle enthaltenen Wörter aufgeteilt. Um typische Bindewörter zu eliminieren, werden nur Wörter mit einer Mindestlänge von fünf Zeichen berücksichtigt.

Für die initiale Tagvergabe wurde ein spezieller Benutzer angelegt welcher diese vornimmt, um jeweils zwischen initialen Tags und Benutzertags unterscheiden zu können. Hierdurch kann vermieden werden die globale Tag-Cloud mit initialen Tags anzufüllen. Die globale Tag-Cloud bietet somit also lediglich ein Abbild der von Benutzern erzeugten Tag-Landschaft.

9. Benutzerverwaltung

Um die im vorherigen Kapitel beschriebenen Benutzerfunktionen zu ermöglichen, ist es notwendig, die Benutzerverwaltung zu realisieren. Die Aufgaben der Benutzerverwaltung von oCOBS wurden bereits in Kapitel 1.4 auf Seite 7 vorgestellt. Ein Benutzer soll sich bei oCOBS registrieren und einloggen können. Nach der Registrierung wird ein persönliches Profil angelegt, welches jederzeit nach einem Login auf einer Profilseite eingesehen und bearbeitet werden kann. Dort können verschiedene profilbezogene Einstellungen geändert werden. Dazu gehört unter anderem das Veröffentlichen oder Verstecken von persönlichen Informationen.

9.1. Lösungswege

Es gibt zwei Möglichkeiten, um die Benutzerverwaltung zu realisieren. Zum einen kann ein geeignetes Framework verwendet werden, das in oCOBS integriert wird. Zum anderen kann auch eine eigene Benutzerverwaltung entwickelt werden. Die nachfolgenden Abschnitte befassen sich mit den verschiedenen Lösungswegen, die hinsichtlich der Bedürfnisse von oCOBS analysiert und bewertet werden. Anschließend werden die Entscheidungen der PG 540 erwähnt. Die Benutzerverwaltung von oCOBS wird anhand dieser Entscheidungen umgesetzt.

9.1.1. Eigene Implementierung

Der erste mögliche Ansatz zur Bereitstellung einer Benutzerverwaltung ist die Entwicklung einer eigenen Lösung. Bei diesem Ansatz muss man sich der Komplexität einer Benutzerverwaltung bewusst sein. Dazu gehört vor allem der Aspekt der Sicherheit. Weitere wichtige Punkte sind das Registrierungsverfahren und der Umfang eines Benutzerprofils. Die folgenden Unterkapitel befassen sich mit den Problemen und möglichen Lösungsansätzen für eine eigene Benutzerverwaltung.

9.1.1.1. Registrierung & Profildaten

Um eine Benutzerverwaltung zu realisieren, muss sich ein Benutzer bei oCOBS registrieren, wobei ein eindeutiger Identifikator, wie z.B. die E-Mail-Adresse, benötigt würde, damit Benutzer voneinander unterschieden werden könnten. Weitere Profildaten könnten während der Registrierungsphase oder später auf der Benutzerprofilseite ausgefüllt werden. Auch eine Bestätigung der Registrierung wäre aus

Sicherheitsgründen (siehe Abschnitt 9.1.1.2) erforderlich. Ein Benutzer, der schon das Registrierungsformular ausgefüllt, aber seinen Account noch nicht durch die E-Mail bestätigt hat, würde als temporärer Benutzer betrachtet und könnte sich noch nicht einloggen.

Identifikator Um Benutzer voneinander unterscheiden zu können, ist die Wahl eines geeigneten eindeutigen Identifikators für den Benutzer zwingend erforderlich. Dieser eindeutige Identifikator lässt sich später nicht mehr ändern. Hier bietet sich beispielsweise die E-Mail-Adresse an. Einerseits ist die E-Mail-Adresse eindeutig und dem Benutzer permanent bekannt. Anderseits kann sie zur Bestätigung der Registrierung, zur Mitteilung von Nachrichten oder für die Anforderung eines neuen Passwortes gebraucht werden. Bei Bedenken bezüglich der Veröffentlichung der E-Mail-Adresse sollte der Benutzer zusätzlich einen Anzeigenamen angeben können. Alternativ könnte auch der von einem Benutzer angegebene Spitzname als Identifikator verwendet werden. Dies ist allerdings nicht optimal, denn einerseits sollte ein Benutzer einen neuen Spitznamen wählen, falls sein bevorzugter Name schon vergeben ist und anderseits dürfte ein Benutzer diesen Namen nicht vergessen, da dieser als Login-Name genutzt würde.

Passwort Es gibt zwei Möglichkeiten zum Erstellen eines Passwortes, entweder ein selbst gewähltes oder ein automatisch generiertes Passwort. Im zweiten Fall würde das automatisch generierte Passwort in der Bestätigungs-E-Mail mitgeschickt und könnte später vom Benutzer geändert werden.

Profildaten Die zusätzlichen Informationen wie z.B. Vorname, Nachname, Institut, Forschungsgebiet wären für oCOBS nur optional. Die Profildaten könnten während der Registrierung vom Benutzer hinzugefügt werden. Falls keine Informationen angegeben werden, könnte zunächst ein leeres Profil angelegt werden, welches auf der Profilseite nach erfolgreichem Einloggen des Benutzers von diesem vervollständigt werden könnte. Dort könnte dieser auch jederzeit festlegen, ob seine persönlichen Informationen öffentlich einsehbar sind oder nicht.

9.1.1.2. Sicherheit

Wie schon genannt muss vor allem der Aspekt der Sicherheit beachtet werden. Folgende Punkte sind besonders zu erwähnen:

Botschutz Ein *Bot* ist ein Computer-Programm, das selbstständig sich wiederholende Aufgaben abarbeitet und versucht, eine Webseite mit unnützen Informationen zu überfluten. Um oCOBS vor Manipulation und Missbrauch durch Bots zu schüt-

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x'; DROP TABLE members; --';
```

Listing 9.1: Beispiel SQL-Injection

zen, muss eine Botschutzfunktion realisiert werden. *CAPTCHA*[1] ist eine mögliche Lösung: Es wird dem Benutzer vor Durchführung einer Aktion eine Aufgabe gestellt, die für Menschen einfach zu lösen ist, den Bot jedoch vor ein Problem stellt. Beispielsweise wird eine zufällige Buchstaben- und Zahlenfolge grafisch verzerrt, die dann vom Benutzer in ein Eingabefeld eingetragen werden muss. Hier stehen viele Open Source Lösungen zur Verfügung (z.B. JCaptcha [26]).

Injection Bei einem Angriff durch SQL-Injection wird ausgenutzt, dass ungeprüfte Werte oder Zeichenfolgen aus einer Benutzereingabe intern an eine SQL-Anfrage übergeben werden. Ein potenzieller Angreifer kann dann die Eingabefelder in einem Formular dazu benutzten, schadhaften SQL-Code in ein System einzuschleusen.

Listing 9.1 zeigt eine beispielhafte SQL-Injection. In der dritten Zeile wird eine wohlgeformte E-Mail-Adresse erwartet. Ohne Überprüfung kann aber auch der String „`DROP TABLE members`“ übergeben werden, wodurch die Tabelle „`members`“ in der Datenbank gelöscht wird ([54] für weitere Informationen). Zum Schutz vor SQL-Injections kann in Java die bereits vorhandene Klasse `PreparedStatement` verwendet werden, die Daten unsicherer Herkunft als getrennte Parameter übergibt und somit schädliche lange SQL-Statements verhindert.

E-Mail-Verifikation Wenn die E-Mail-Adresse als eindeutiger Identifikator für Benutzer verwendet wird, muss eine E-Mail-Verifikation durchgeführt werden. Die Eindeutigkeit einer E-Mail-Adresse beugt Benutzerdoubletten in der Datenbank vor. Zudem wird ein Syntaxcheck der E-Mail-Adresse benötigt, der durch Verwendung von vorgefertigten Skripts oder einer BNF[2] ermöglicht, die Wohlgeformtheit einer E-Mail-Adresse zu überprüfen. Die Bestätigungs-E-Mail wird benötigt um sicherzustellen, dass diese E-Mail-Adresse tatsächlich vorhanden ist und der mit dieser E-Mail-Adresse registrierte Benutzer auch der Inhaber dieser Adresse ist.

Verschlüsselung der Passwörter Die Verschlüsselung der Passwörter kann durch eine Einweg-Hashfunktion realisiert werden. Eine Hashfunktion generiert aus einer beliebig großen Eingabe einen String fester Länge als Ausgabe, wobei aus der Ausgabe nicht auf die Eingabe rückgeschlossen werden kann. In der Datenbank wird nur der Hashwert des Passwortes gespeichert, aus dem niemand das ursprüngliche

[1] Completely Automated Public Test to tell Computers and Humans Apart
[2] Backus-Naur-Form

Passwort ableiten kann. Das Passwort wird wie folgt überprüft: Ein Vergleich des Hashwerts aus der Datenbank mit dem Hashwert des eingegebenen Passwortes beim Login muss die Gleichheit beider Werte zeigen, um einen Benutzer zu verifizieren. SHA1[3] ist beispielsweise eine sichere und einfache Lösung für die Bildung solcher Hashwerte.

9.1.1.3. Vor- und Nachteile

Eine eigene Implementierung der Benutzerverwaltung hat den großen Vorteil, dass diese genau an die Bedürfnisse von oCOBS und deren Nutzer angepasst werden kann. Alle Funktionen können genau wie geplant umgesetzt werden, ohne Kompromisse eingehen zu müssen.

Der wesentliche Nachteil einer eigenen Implementierung ist die große Komplexität. Viele Aspekte müssen beachtet werden, daher entsteht es ein hoher Implementierungsaufwand bei der Umsetzung. Das Registrierungsverfahren inklusive Filterung gültiger Eingaben und Verifikation durch Bestätigungs-E-Mail beinhaltet zusätzlichen Aufwand. Des Weiteren müssen auch Passwörter abgespeichert und abgesichert werden.

9.1.2. LDAP

Ein weiterer Lösungsansatz für die Realisierung der Benutzerverwaltung für oCOBS ist das *Lightweight Directory Access Protocol (LDAP)* [61]. LDAP ist der De-facto-Standard für Autorisierung, Authentifizierung und Benutzerverwaltung. Er findet z.B. Verwendung in Adressbüchern wie Microsoft Outlook und Mozilla Thunderbird.

Durch das LDAP-Protokoll lässt sich ein Verzeichnisdienst realisieren, bei dem Benutzeranfragen durch das Protokoll an den Verzeichnisserver, der für die Ausführung der Operationen verantwortlich ist, weitergeleitet werden. Der Verzeichnisserver nutzt als Datenstruktur einen hierarchischen Baum.

Für die Realisierung der Benutzerverwaltung von oCOBS würde dies bedeuten, dass oCOBS-Nutzer als LDAP-Objekte im Verzeichnis gespeichert werden. Diese können eindeutig mit dem *Distinguished Name (DN)* identifiziert werden. Der DN setzt sich aus verschiedenen Typen, den *Relative Distinguished Names (RDN)* zusammen und kann somit mit Attributen gefüllt werden. Beispiele für RDNs sind `UID = User ID`, `cn = Common Name` und `sn = Surname`. Für oCOBS bedeutet dies, dass Informationen, die der Nutzer über seine eigene Person angeben möchte, als RDNs zu seinem Distinguished Name hinzugefügt werden. Auch andere Attribute, wie die verfassten Tags des Nutzers können als RDNs gespeichert werden. Die einfache Struktur und die hohe Verfügbarkeit von freien Frameworks und Tools für die Realisierung von LDAP sind als Hauptvorteile zu nennen.

[3] Secure Hash Algorithm 1

Zieht man allerdings die schon vorhandene Architektur von oCOBS in Betracht, gibt es aber auch einige Nachteile, die sich durch die Benutzung von LDAP ergeben würden. Wäre der Datenbestand von oCOBS, also die Publikationen und Autoren mit ihren Attributen, bereits in einem hierarchischen Baum gespeichert, könnten diese Daten mit wenig Aufwand mit den neuen Benutzerdaten verknüpft werden. Da der Datenbestand von oCOBS in einer relationalen Datenbank verwaltet wird, kann dieser positive Effekt von LDAP nicht genutzt werden. Eine nachträgliche Umwandlung der Datenbank in eine hierarchische Verzeichnisstruktur wäre sehr zeitintensiv und hat den Nachteil, dass Schreibzugriffe, vor allem bei einem so hohen Datenbestand wie in oCOBS, in hierarchischen Datenbanken langsam sind und es zu Performanceproblemen bei der Ausführung von komplexen Suchanfragen kommen kann.

Sollte LDAP als Benutzerverwaltung für oCOBS genutzt werden, so müssten also zwei Datenbanken, eine relationale (historisch bedingt) und eine hierarchische (LDAP), parallel verwaltet werden. Im Hinblick auf den so entstehenden Implementierungs- und Anpassungsaufwand für viele der neuen Funktionen hat sich die Projektgruppe gegen die Verwendung von LDAP zur Unterstützung der Benutzerverwaltung entschieden.

9.1.3. OpenID

9.1.3.1. Grundlagen

OpenID [35] ist ein dezentrales Authentifizierungssystem im Internet. Benutzer müssen sich bei einem OpenID-Provider nur einmalig registrieren, um dann verschiedene Webseiten, die OpenID unterstützen, direkt nutzen zu können. Zur Authentifizierung wählt der Benutzer beim Provider eine eindeutige OpenID (eine URL) und ein Passwort, mit denen er sich dann bei OpenID-unterstützenden Systemen anmelden kann. Bekannte Provider sind Google, AOL oder Yahoo, die registrierten Benutzern schon implizit die OpenID-Funktionalität anbieten. Es existieren aber auch viele Anbieter, wie z.B. myOpenID [31], bei denen Benutzer einen OpenID-Account beantragen können, ohne an weitere Funktionalität der Provider gebunden zu sein.

Die Grundidee von OpenID ist das Konzept *Single-Sign-On* (SSO), welches erlaubt, dass der Benutzer nach einer einmaligen Authentifizierung von anderen OpenID-Systemen als authentifizierter Benutzer erkannt wird. Die OpenID-Technologie besteht aus vier Komponenten: *End User*, *Identifier*, *Identity Provider* (IdP) und *Relying Party* (RP). Die Idee ist, dass der Benutzer (End User) sich bei einem existierenden OpenID-Anbieter (Identity Provider) registrieren kann und einen OpenID-Account (Identifier) erhält. Mit diesem Account kann der Benutzer sich dann auf jeder Internetseite (Relying Party), die OpenID unterstützt, direkt einloggen. Der Ablauf der Authentifizierung mittels OpenID besteht aus folgenden sechs Schritten:

1. Der Benutzer gibt seine OpenID auf der Webseite ein.
2. Eine Verbindung zwischen der Webseite und dem Anbieter (Provider) wird hergestellt.
3. Der Benutzer wird von der Webseite zum Anbieter umgeleitet.
4. Der Benutzer authentifiziert sich beim Anbieter (z.B. durch Passworteingabe).
5. Der Benutzer wird mit einer Signatur vom Anbieter zurück zur Webseite geleitet.
6. Die Webseite verifiziert diese Signatur.

9.1.3.2. Vor- und Nachteile

Der schon genannte große Vorteil von OpenID ist, dass ein Benutzer sich nicht auf jeder Webseite neu registrieren muss und sich somit nicht jedesmal neue Authentifizierungsdaten merken muss. Neben reinen OpenID-Anbietern wird OpenID auch von vielen populären Internetkonzernen wie z.B. Google und Yahoo implizit unterstützt. Aus diesem Grund sind viele Internetnutzer schon mit einem OpenID-Account ausgestattet. Sie können sich deswegen ohne Registrierung direkt bei oCOBS authentifizieren, wenn oCOBS OpenID unterstützt. Aufgrund der Vielzahl verschiedener Provider ist auch das Erstellen eines neuen OpenID-Accounts kein Problem.

Die Eindeutigkeit des OpenID-Accounts ist ebenfalls ein Vorteil, da durch die eindeutige Identifizierung eines Benutzers Benutzerdoubletten leicht vermieden werden können.

Ein weiterer Vorteil ist, dass die Registrierung, die Authentifizierung und die Speicherung der Passwörter nicht von oCOBS umgesetzt werden müssen, was erhebliche zeitliche Einsparungen bei der Umsetzung im Vergleich zu einer eigenen Implementierung zur Folge hat.

OpenID bringt zudem noch weitere nützliche Funktionalitäten, wie „OpenID Attribute Exchange“ [34], mit sich: Die Relying Party kann bestimmte Attribute eines Benutzers direkt vom Provider auslesen, sofern der Benutzer die Informationen als sichtbar eingestellt hat. Solche Attribute sind z.B. E-Mail-Adresse, Vorname, Nachname usw. Eine andere wichtige Funktionalität von OpenID ist „OpenID UI Extension“ [36]: Mittels dieser Erweiterung ist es möglich, *LoginWithoutLeavingPage* zu realisieren. Das bedeutet, dass die Kommunikation zwischen dem Benutzer und dem OpenID-Anbieter nur innerhalb eines Popup-Fensters stattfindet. Die ursprüngliche Webseite, die der Benutzer besucht hat, bleibt geöffnet.

Es gibt allerdings auch einige Nachteile, die zu beachten sind. Da OpenID Single-Sign-On (SSO) verwendet, ist ein Benutzer der sich einmal bei einem OpenID Provider einloggt auch bei allen anderen Webseiten eingeloggt, die er mittels OpenID verwendet. Ein Logout vor dem Verlassen der Webseite ist insbesondere zwingend

notwendig, wenn ein öffentlicher PC zum Besuchen der Seite verwendet wird. Aufgrund des SSOs entsteht ein weiterer Nachteil: Wird der OpenID Provider infiltriert, hat ein Angreifer direkten Zugriff auf alle der von betroffenen Benutzer verwendeten Webseiten, welche OpenID als Authentifizierung unterstützen. Das gleiche Problem entsteht bei unsicher gewählten Passwörtern.

Ein weiterer Nachteil ist das mangelnde Vertrauen, das Benutzer der OpenID-Technologie entgegen bringen könnten: Da die Authentifizierungsdaten bei einem externen Anbieter liegen und nicht bei den Webseiten, die besucht werden sollen, muss einem Provider ein großes Maß an Vertrauen entgegen gebracht werden. Der Ausfall eines OpenID-Anbieters kann ebenfalls ein potentielles Problem darstellen.

9.1.3.3. Java-Framework

Die folgenden vier Java-Frameworks wurden von der PG untersucht:

- JOpenID [29]
- JOID [28]
- OpenID4Java [37]
- dyuproject [64]

JOpenID ist eine reine Open-Source-Lightweight-Implementierung für OpenID 2.0, während JOID nicht nur OpenID 2.0 unterstützt sondern auch 1.x. Beide Implementierungen sind allerdings nicht gut dokumentiert. OpenID4Java und dyuproject sind dagegen zwei Implementierungen für OpenID 2.0 und 1.x, die einerseits sehr gut dokumentiert sind und anderseits viele Erweiterungen, wie z.B. die schon genannten „OpenID Attribute Exchange“ und „OpenID UI Extension“ unterstützen (siehe Abbildung 9.1). Des Weiteren stehen viele Beispielcodes und entsprechende Erklärungen zur Verfügung.

	License	Relying Party	Identity Provider	Compatibility	Extension
JOpenID	Apache v2	Ja	Nein	2.0	Nein
dyuproject	Apache v2	Ja	Nein	2.0, 1.x	Ja
JOID	Apache v2	Ja	Ja	2.0, 1.x	Nein
OpenID4Java	Apache v2	Ja	Ja	2.0, 1.x	Ja

Abbildung 9.1.: Vergleich der OpenID-Frameworks

9.1.4. Entscheidungen

Da LDAP, aufgrund des hohen Implementierungs- und Anpassungsaufwands, zur Realisierung einer Benutzerverwaltung im vornherein ausgeschlossen wurde (vgl. Abschnitt 9.1.2 auf Seite 65), bleibt eine OpenID-basierte Lösung oder eine eigene Implementierung zur Auswahl.

Die wesentlichen Vor- und Nachteile der beiden Lösungswege wurden bereits in Abschnitt 9.1.1.3 auf Seite 65 und Abschnitt 9.1.3.2 auf Seite 67 erwähnt. Zur Wahl einer passenden Lösung für oCOBS wurden die beiden Möglichkeiten direkt miteinander verglichen und schließlich von der PG entschieden, dass die Benutzerverwaltung mit OpenID realisiert werden soll. Der ausschlaggebende Aspekt für das Verwenden von OpenID ist dabei, dass der Arbeitsaufwand für die Implementierung deutlich geringer ist, als der Arbeitsaufwand für eine eigene Benutzerverwaltung. Auch die Benutzerfreundlichkeit von OpenID und die vielfältige Nutzbarkeit der Accounts sprechen für OpenID. Es wurde weiterhin beschlossen, dass das Open-Source-Java-Framework „dyuproject“ zur Realisierung genutzt wird, da „dyuproject“ entgegen anderen Frameworks *LoginWithoutLeavingPage* unterstützt und die oCOBS-Nutzer somit die ursprüngliche Webseite nicht verlassen müssen.

Da OpenID nur die Registrierung, die Authentifizierung sowie das Ein- und Ausloggen übernimmt, müssen neben der Integration des OpenID-Frameworks sämtliche oCOBS-spezifische Funktionen dennoch selbst implementiert werden. Die konkreten eigenen Umsetzungen sind die Folgenden:

- **Login- und Logout-Block**
  Auf der linken Seite der Web-Oberfläche sollte sich der Login- und Logout-Block befinden, damit sich ein Benutzer jederzeit ein- und ausloggen kann. Hat sich der Benutzer noch nicht authentifiziert, sollten hier ein Link zur Hilfsseite und ein Login-Button zur Verfügung gestellt werden, welcher ein Popup-Fenster aufruft, in dem der Benutzer sich bei OpenID einloggen kann. Ist der Benutzer bereits eingeloggt, sollten in diesem Block ein Logout-Button zum Abmelden und der OpenID-Account des Benutzers erscheinen, welcher zur Profilseite verlinkt ist.

- **Blankoprofil und Benutzerprofil**
  Nach dem ersten Einloggen eines Benutzers sollte automatisch ein Blankoprofil generiert werden. oCOBS kann danach die E-Mail-Adresse und Vor- und Nachname des Benutzers von dessen OpenID-Anbieter einlesen, falls diese Attribute öffentlich sichtbar sind. Auf der Profilseite würden dann die aktuellen Profildaten des authentifizierten Benutzers angezeigt werden. Diese Daten könnten vom Benutzer jederzeit aktualisiert werden. Außerdem könnte die E-Mail-Adresse auf „öffentlich“ oder „nicht öffentlich“ geschaltet werden. Dies hätte zur Folge, dass andere Nutzer diese auf dem öffentlichen Profil des Benutzers einsehen können oder nicht. Die öffentliche Profilseite eines Benutzers

wäre durch den Klick auf den „Verfasser“ eines Kommentars oder eines Tags zu einer Publikation aufrufbar.

- **Sperrung von OpenID**
  Die Admin-Oberfläche sollte um eine Funktion zur Sperrung von OpenID-Accounts erweitert werden. Dort könnte ein OpenID-Account durch Änderung des Status gesperrt werden. Der Benutzer, dessen OpenID-Account gesperrt wird, könnte sich danach nicht mehr bei oCOBS einloggen. Die zugehörigen Tags und Kommentare dieses Benutzers sollten ebenfalls gesperrt werden.

9.2. Realisierung

Neben der Integration des OpenID-Frameworks wurde die Benutzerverwaltung ähnlich wie bei der Bewertung und dem Kommentar auch mit Hilfe von JavaScript, AJAX, jQuery und JSON realisiert.

9.2.1. Implementierung der Benutzerverwaltung

Login- und Logout-Block

Zunächst wurde eine Java Bean angelegt, um jederzeit überprüfen zu können, ob sich ein Nutzer von oCOBS bereits angemeldet hat. Nicht nur die Gestaltung für den Login- und Logout-Block, sondern auch viele andere Teile hängen von dieser Überprüfung ab.

Der Login- und Logout-Block wird in allen Seiten eingebaut, damit der Benutzer sich jederzeit an- oder abmelden kann. Bei der Implementierung des Ein- und Ausloggens muss das OpenID-Framework integriert werden. Ein Servlet beobachtet die Kommunikation zwischen dem Benutzer und dem OpenID-Anbieter, um den OpenID-Account und die zugehörigen lesbaren Informationen zu erhalten. Sobald diese Kommunikation abgeschlossen ist, wird die Behandlung des Loginvorgangs an ein anderes Servlet weitergeleitet. Um das Feature *LoginWithoutLeavingPage* umzusetzen, wird neben den notwendigen JSP-Seiten auch eine statische Seite verwendet, welche das Popup-Fenster schließt und den Loginvorgang fortsetzt.

Benutzerprofilseite

Ein Servlet kümmert sich um die Anfragen nach Profilseiten. Dabei kann ein optionaler Parameter „id“ übergeben werden. Mit diesem Parameter kann ein Benutzer immer eindeutig eine Profilseite aufrufen. Es ist auch möglich, seine eigene Profilseite durch Weglassen dieses Parameters aufzurufen. Ein anderes Servlet und eine Menge von JavaScript-Funktionen kümmern sich um die Aktualisierung der Profildaten.

Zuerst müssen die Änderungen validiert werden. Dabei gibt es zwei Voraussetzungen: Zum einen muss die eingegebene E-Mail-Adresse wohlgeformt sein und zum anderen muss der ausgewählte Anzeigename sichtbar sein. Wenn diese Voraussetzungen erfüllt sind, werden die Änderungen anschließend per AJAX-Interaktion an den Server gesendet und dann aktualisiert.

9.2.2. Web-Oberfläche der Benutzerverwaltung

Login- und Logout-Block

Der Login- und Logout-Block wurde anhand vorhergehender Entscheidungen (siehe Abschnitt 9.1.4 auf Seite 69) implementiert. Zunächst stehen zwei Menüpunkte zur Auswahl: Durch einen Klick auf einen Link wird ein Benutzer zu einer Hilfeseite umgeleitet. Ein Benutzer, welcher noch keinen OpenID-Account besitzt, kann dort einen OpenID-Anbieter wählen, um einen OpenID-Account zu beantragen. Durch einen Klick auf den Login-Button erscheint ein Popup-Fenster (siehe Abbildung 9.2). Dort kann der Benutzer seinen OpenID-Account direkt oder mit Hilfe von dem eingebauten „OpenID-Selector“ eingeben. Danach wird eine Verbindung zwischen oDOBS und entsprechenden OpenID-Anbieter aufgebaut, damit der Benutzer sich bei dem Anbieter authentifizieren kann. Falls ein Benutzer sich nicht bei dem Anbieter authentifizieren kann oder der OpenID-Account bei oDOBS schon gesperrt ist, wird eine Fehlermeldung angezeigt. Sonst werden der Logout-Button zum Ausloggen und der OpenID-Account-Name angezeigt. Mit einem Klick auf den OpenID-Account wird der Benutzer zur eigenen Profilseite umgeleitet.

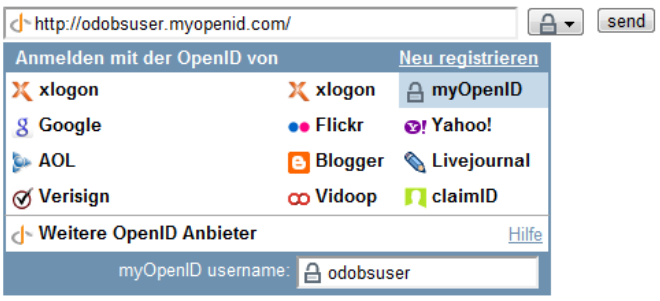

Abbildung 9.2.: Popup-Fenster für Einloggen

Profilseite

Die Profilseite wird nur angezeigt, wenn ein Benutzer sich erfolgreich angemeldet hat. Auf jeder Profilseite wird auch eine Benutzer-Tag-Cloud angezeigt (siehe Abschnitt 8.3.3 auf Seite 58). Möchte der Benutzer das Profil eines gesperrten Benutzers anzeigen, sieht er nur eine Meldung, dass dieser Benutzer gesperrt wurde. Zur eigenen Profilseite (siehe Abbildung 9.3) gelangt der Benutzer entweder mit einem Klick auf den Link im Login- und Logout-Block oder einem Klick auf den Anzeigenamen bei den eigenen Kommentaren und Tags. Auf dieser Seite kann der Benutzer jederzeit seine persönlichen Informationen, ausgenommen des OpenID-Accounts, ändern. Zudem kann er auch seinen Anzeigenamen für verfasste Kommentare wählen und die Sichtbarkeit von E-Mail-Adresse, Namen oder Nicknamen einstellen.

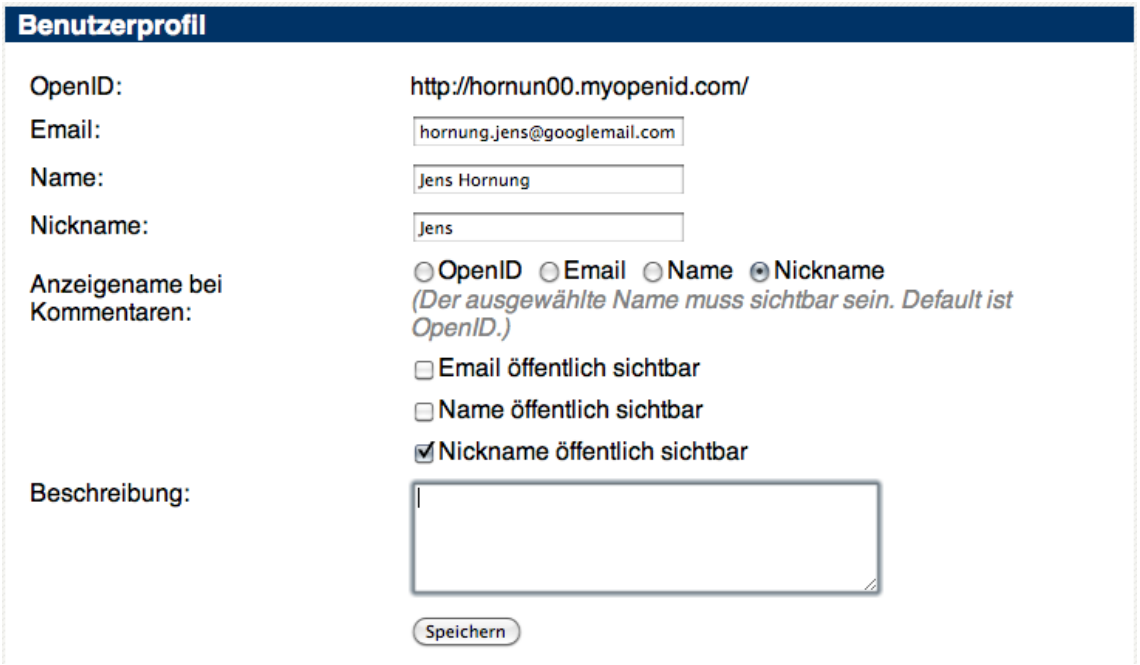

Abbildung 9.3.: Eigenes Benutzerprofil

Die öffentliche Profilseite für andere Benutzer (siehe Abbildung 9.4 auf der nächsten Seite) kann von allen angemeldeten Benutzern mit einem Klick auf den Anzeigenamen bei Kommentaren und Tags aufgerufen werden. Der OpenID-Account und die Beschreibung werden immer angezeigt. Andere Profildaten werden nur angezeigt, falls sie vom Benutzer als sichtbar gekennzeichnet wurden.

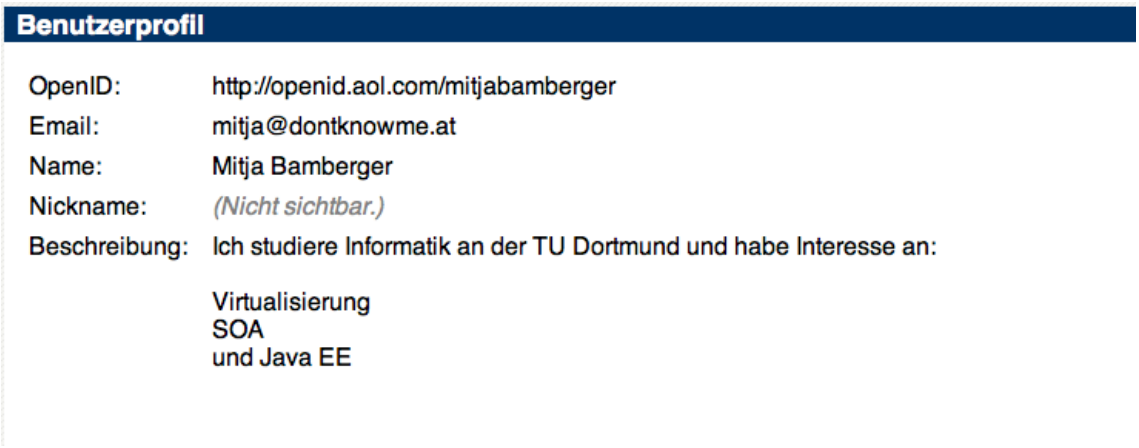

Abbildung 9.4.: Benutzerprofil von anderen Benutzern

10. Import

Aufgrund der Ergebnisse der von der Projektgruppe durchgeführten Analyse des bisherigen Imports in Kapitel 7 auf Seite 40 hat die PG sich zu einem kompletten Redesign der Importfunktion von oCOBS entschieden. Ausschlaggebend war zum einen die schlechte Performance, und zum anderen die mangelhafte Stabilität. Dies soll in einer neuen Version des Imports zu verbessert werden.

10.1. Identifikation der Anwendungsfälle

Bei der Auswertung des Datenimportprozesses hat die Projektgruppe die in Tabelle 10.1 sichtbaren Anwendungsfälle identifiziert. Die „Datenquelle“ gibt an, in welchem Format die Daten vorliegen. Bei „Datenart“ wird unterschieden, ob es sich um Publikations- oder Kategoriedaten (siehe Kapitel 7.1.1 auf Seite 40 und Kapitel 7.1.2 auf Seite 42) handelt, während „Quelle“ angibt, ob der Datensatz lokal vorliegt oder zunächst vom Server der DBLP [59] heruntergeladen werden soll. Der BibTex-Import soll zusätzlich zu dem Import einer lokalen Datei noch den Import eines Eingabestrings im BibTex-Format realisieren. Die Importart ist die einzige wesentliche Neuerung, welche die Projektgruppe mit dem Redesign des Imports realisieren will. Es wird unterschieden zwischen vollständigem Import und Datenupdate. Die letzte Spalte gibt die geschätzte Häufigkeit an, mit welcher der jeweilige Vorgang ausgeführt wird. Weiterhin werden in dieser Spalte auch Vorgänge markiert, die lediglich zu Testzwecken dienen. Dadurch ist es leicht zu bestimmen, an welcher Stelle Verbesserungen möglichst große Auswirkungen auf das Gesamtsystem haben.

Datenquelle	Datenart	Quelle	Importart	Häufigkeit
DBLP	Publikationen	Online	vollständig	einmalig
DBLP	Publikationen	Lokale Datei	vollständig	Testzwecke
DBLP	Publikationen	Online	Datenupdate	sehr häufig
DBLP	Publikationen	Lokale Datei	Datenupdate	Testzwecke
DBLP	Kategorien	Online	vollständig	sehr häufig
DBLP	Kategorien	Lokale Datei	vollständig	Testzwecke
BibTex	Publikationen	Lokale Datei	zusätzliche Daten	hin und wieder
BibTex	Publikationen	Eingabestring	zusätzliche Daten	hin und wieder

Tabelle 10.1.: Varianten der neuen Importfunktion

10.1.1. DBLP-Publikationsimport

Der Publikationsimport ist in der derzeitigen Version des Imports der größte Verursacher langer Laufzeiten. Grund dafür ist der stetig anwachsende Datenbestand der DBLP, welcher den Importvorgang immer langsamer macht. Da der Import für jeden Eintrag zunächst den aktuellen Datenbestand daraufhin durchsucht, ob die aktuell zu importierende Publikation bereits erfasst ist, steigert sich die Laufzeit des Imports quadratisch mit der Anzahl der Datensätze in der `DBLP.xml`.

Aus diesem Problem heraus ergibt sich für die Projektgruppe die erste wichtige Änderung. Es wird in Zukunft bei DBLP-Publikationsimporten zwischen einem vollständigen Import (*vollständig*) und einem Teilimport (*Datenupdate*) unterschieden.

Vollständiger Import

Ein vollständiger Import ist in gewisser Weise ein Sonderfall, der lediglich dann auftritt, wenn eine neue oCOBS-Instanz aufgesetzt wird oder aufgrund eines Datenverlustes alle Daten neu eingepielt werden müssen. Hierbei unterscheiden wir wie im alten Import die Möglichkeiten eines Online- und eines lokalen Imports. Im Normalfall wird ein vollständiger Import direkt mit den aktuellen Online-DBLP-Daten gestartet, die automatisch heruntergeladen und temporär gespeichert werden. Allerdings ist es durchaus denkbar, dass zu Testzwecken modifizierte lokale DBLP-Publikationsdaten eingespielt werden sollen. Daher hat sich die Projektgruppe dazu entschieden, den Import von DBLP-Daten aus einer lokal vorliegenden Datei weiterhin zu unterstützen.

Inkrementeller Import

Auf Grund der gewünschten Performance-Erhöhung soll in regelmäßigen Abständen ein Teilimport (ein Update der Publikationsdaten) stattfinden, der deutlich effizienter ist als ein regelmäßiger vollständiger Importvorgang, wie er derzeit ausgeführt wird. Bei einem solchen Teilimport werden alle Daten der DBLP, die seit dem letzten Importvorgang neu hinzugekommen sind, identifiziert und in oCOBS importiert. Die Schwierigkeit hierbei ist, zu identifizieren, bei welchen Daten der DBLP es sich um neue Daten handelt, die noch nicht von oCOBS erfasst wurden. Hierbei kann der Syntax der DBLP-Daten behilflich sein (siehe Kapitel 7.1.1 auf Seite 40). Listing 10.1 stellt einen zufällig ausgewählten Eintrag der `DBLP.xml` dar. Das Attribut `mdate` gibt das Datum, an dem dieser Eintrag zuletzt geändert wurde, im `ISO 8601`-Format an [11]. Im gegebenen Beispiel wäre also der Datensatz zuletzt am 18.8.2004 modifiziert worden.

Durch die Information, wann ein Publikationseintrag zuletzt modifiziert worden ist, lässt sich leicht ermitteln, ob dieser bereits in den oCOBS-Datenbestand übernommen wurde. Hierfür ist es notwendig, dass das Datum des letzten erfolgreichen Imports im System gespeichert wird. Danach kann beim folgenden Teilimport für

```
<inproceedings mdate="2004-08-18" key="conf/adc/DongB04">
<author>Ce Dong</author>
<author>James Bailey</author>
<title>Static Analysis of XSLT Programs.</title>
<pages>151-160</pages>
<year>2004</year>
<crossref>conf/adc/2004</crossref>
<booktitle>ADC</booktitle>
<ee>http://crpit.com/confpapers/CRPITV27Dong.pdf</ee>
<url>db/conf/adc/adc2004.html#DongB04</url>
</inproceedings>
```

Listing 10.1: Eintrag in der DBLP.xml

jeden Datensatz zunächst abgeglichen werden, ob er seit dem letzten Import modifiziert worden ist. Dies geschieht durch den Vergleich der beiden Datumswerte, also dem `mdate` des aktuellen Datensatzes und das im System gespeicherte Datum des letzten Imports. Ist dies nicht der Fall, so braucht er nicht berücksichtigt zu werden. Die Laufzeit der Überprüfung ist lediglich linear abhängig von der Anzahl der enthaltenen Datensätze und reduziert die Anzahl der Überprüfungen, ob die Publikation bereits vorhanden ist, enorm. Die Laufzeit des folgenden eigentlichen Importvorgangs kann durch regelmäßige Importe gering gehalten werden.

10.1.2. DBLP-Kategorieimport

Auch beim DBLP-Kategorieimport ist eine Unterscheidung zwischen einem Komplettimport (*Komplett*) und einem Teilimport (*Datenupdate*) in Betracht zu ziehen, denn auch hier ist ein Komplettimport eher ein initialer Vorgang, der nur einmalig beim Aufsetzen einer neu erstellten oCOBS-Instanz auftritt. Viel häufiger besteht hingegen der Bedarf, den Datenbestand zu aktualisieren.

Im Gegensatz zum Datensatz der `DBLP.xml` existiert für die in der `DBLP_bht.xml` gespeicherten Kategoriedaten kein Attribut, welches das Datum der letzten Änderung des Eintrags angibt (vgl. Listing 10.2 auf der nächsten Seite) [11]. Für eine Optimierung des DBLP-Kategorieimports wäre es daher notwendig, einen syntaktischen Vergleich zwischen zwei `DBLP_bht.xml`-Dateien durchzuführen, um geänderte Datensätze herauszufiltern.

Aufgrund der Tatsache, dass aus Sicht der Projektgruppe die größten Performance-Probleme beim Publikationsimport vorhanden waren, hat sich diese gegen eine Neuimplementierung des Kategorieimports entschieden.

10.1.3. BibTex-Import

Zusätzlich bietet oCOBS die Funktion, Daten im BibTex-Format zu importieren. BibTex-Dateien sind eher Nebenquellen; Hauptdatenquelle von oCOBS ist wie gehabt die DBLP. Weiterhin sind die zu importierenden BibTex-Dateien eher klein im

```
<bht key="/db/journals/cacm/ButterworthOS91.bht" title="...">
<cite key="journals/cacm/ButterworthOS91" style="sigmoddlcitations"/>
<abstract copyright="ACM" year="1991">
 GemStone, a commercially avatiable object database management system(ODBMS), ...
<hr/>
<cite key="journals/cacm/ButterworthOS91" style="cdrom"/>
<h2>
Online Edition: <a href="http://www.acm.org/dl/">
ACM Digital Library</a>
</h2>
<a href="http://www.acm.org/pubs/citations/journals/cacm/1991-34-10/p64-butterworth/">
Citation Page</a>
<hr/>
<cite key="journals/cacm/ButterworthOS91" style="citations"/>
<footer style="acm" name="CACM"/>
</bht>
```

Listing 10.2: Eintrag in der DBLP_bht.xml

Vergleich zur `DBLP.xml`. Der BibTex-Import wird also unregelmäßig und mit eher geringem Datenumfang ausgeführt. Durch die eher geringe Laufzeit eines BibTex-Imports, welche der geringen zu importierenden Datenmenge geschuldet ist, und der fehlenden Optimierungsmöglichkeiten kann der alte BibTex-Import beibehalten werden.

10.2. Anpassungen beim Redesign

Durch das Redesign ergibt sich sowohl die Notwendigkeit, als auch die Möglichkeit ein paar Anpassungen zu machen. Die folgenden Kapitel beschreiben diese.

JavaABC

Wie in Kapitel 7.2 auf Seite 43 beschrieben wird, sind die Anforderungen an den Datenimport kaum Veränderungen unterworfen, da das Format der DBLP sehr stabil ist. Dadurch entfällt der große Vorteil, den JavaABC bietet. Weiterhin stellte sich heraus, dass der Import Laufzeitprobleme aufweist. Um diese zu beheben ist von Hand optimierter Code besser geeignet als automatisch generierter. Daher haben wir bei der Neuimplementierung der Importfunktion auf JavaABC verzichtet.

Parser

Für die in Kapitel 7.1.1 auf Seite 40 beschriebenen APIs existieren verschiedene Implementierungen. Ursprünglich sollte die Standard-Implementierung von Sun Verwendung finden, jedoch kam es dabei aufgrund eines bekannten Fehlers zu einem Programmabbruch wegen Mangel an freiem Arbeitsspeicher. Der Wechsel zur *Woodstox*-Implementierung [52] beseitigte dieses Fehlverhalten.

Woodstox hat zudem den Vorteil, dass es die einzige Implementierung ist, die den (noch experimentellen) *StAX2* Standard umsetzt. Von den neuen Funktionen wird u.a. die Methode `XMLStreamReader.skipElement()` verwendet.

Klassenmodell

Die Projektgruppe hat im Zuge des Redesigns der Importfunktion einige Änderungen am zugehörigen Klassenmodell durchgeführt. Grundgedanke ist zwar immer noch eine logische Trennung der verschiedenen Importarten, jedoch sollen gemeinsame Basisfunktionalitäten zentral vorliegen. Ausgangspunkt hierfür ist eine abstrakte Klasse, welche eben diese Basisfunktionalitäten bietet, die von sämtlichen Importvarianten benötigt wird. Ein Beispiel wäre die Etablierung einer Datenbankverbindung, um die eingelesenen Daten zu speichern. Von dieser Klasse wird jeweils eine Klasse für jede Importvariante abgeleitet, welche dann weitere spezielle Funktionalitäten bietet.

Logging

Teil der Entwicklung durch die *Projektgruppe 513: Dortmund Online Public Access Catalog* war die Implementierung einer umfangreichen Loggingfunktion. Dies sollte dazu dienen, eventuelle laufzeittechnische Schwachstellen der Implementierung aufzudecken und zu beheben. Problematisch ist es allerdings, wenn die Loggingfunktion selbst mitverantwortlich für die schlechte Performance der Importfunktion ist.

Die Loggingfunktion erfasst sehr feingranular sämtliche durch die Importversion durchgeführten Änderungen. Dadurch entstehen sehr große Logdateien mit einer Größe von mehreren Gigabyte, weshalb die Logdateien ihren eigentlichen Zweck nicht erfüllen können, da aufgrund des gewaltigen Umfangs jegliche Übersicht fehlt.

Der neue Import macht aus diesem Grund keinen übermäßigen Gebrauch von der Loggingfunktion. Stattdessen werden nur die notwendigsten Ausgaben erzeugt, auf Datenbankzugriffe, die nur statistischen Auswertungen dienen, wird vollständig verzichtet.

Initiale Tagvergabe

Der Import wurde um eine Funktion ergänzt, die jeder importierten Publikation initiale Tags zuweist. Die genaue Vorgehensweise wird in Kapitel 8.3.3 auf Seite 61 beschrieben.

Internet-Zugriffe

Der alte Import hat beim Bearbeiten jeder in der `dblp.xml` vorkommenden Person eine Internetverbindung zum Server der DBLP aufgebaut, um die dort hinterlegte

Personenseite aufzurufen und bestimmte Informationen aus dem HTML-Quelltext zu parsen.

Dies hat den Import deutlich verlangsamt, denn alle auf den Internetseiten der DBLP vorhandenen Informationen sind auch in der `dblp.xml` vorhanden (die Internetseiten werden aus dieser Datei erzeugt). Der neue Import durchläuft aus diesem Grund die XML-Datei mehrmals. In einem ersten Schritt werden nur Personeneinträge importiert. Anschließend erfolgt ein zweiter Durchlauf, in dem Publikationseinträge importiert werden. Dies hat zur Folge, dass die bei einer Publikation genannten Autoren schon in der Datenbank vorhanden sind, und so können direkt die entsprechenden Referenzen auf die Datensätze erfolgen.

Der mehrmalige Durchlauf der Datei hat kaum negative Folgen für die Geschwindigkeit, denn sofern nur syntaktische Überprüfungen – wie in diesem Fall – vorgenommen werden, ist die zusätzliche Laufzeit mit ca. einer halben Minute anzusetzen.

DBLP-Fehler

Während des Testens des neuen Imports kam es immer wieder zu Problemen, die durch Fehler in der `dblp.xml` hervorgerufen wurden. Meist schien es sich dabei um Tippfehler zu handeln, die der manuellen Erfassung der Einträge bei der DBLP zuzuschreiben waren. Die Fehlersuche gestaltete sich äußerst schwierig, weil durch die langen Laufzeiten des Imports immer wieder Verzögerungen entstanden sind. Letztendlich wurde mit den fehlerhaften Einträgen so verfahren, dass den entsprechenden Publikationen, in denen sie vorkommen, ein bestimmter Status zugewiesen wird, der dem Administrator ein schnelles Auffinden ermöglicht (diese Funktion wurde vom alten Import übernommen). Im Anschluss können die Einträge dann manuell überprüft werden.

10.3. Fazit

Durch den neuen Publikationsimport konnte in weniger als einer Minute der komplette Inhalt der `DBLP.xml` auf aktualisierte Datensätze untersucht werden. Dies führte dazu, dass sich die Anzahl der zu importierenden Datensätze erheblich reduzierte und die Laufzeit der Importfunktion drastisch reduziert werden konnte. Dadurch besitzt oCOBS einen sehr performanten Teilimport, der die Möglichkeit bietet, regelmäßige Updates durchzuführen, um die Datenqualität von oCOBS durch aktuelle DBLP-Daten zu verbessern.

11. Suche

Bei der Weiterentwicklung der Suche gab es mehrere offene Punkte, die es zu erledigen galt. Zum Einen war dort das Speicherleck, das bei der im Kapitel 5 auf Seite 33 durchgeführten Analyse gefunden wurde. Dieses musste behoben werden, bevor weitere Änderungen durchgeführt werden konnten. Danach mussten Erweiterungen an der Suche vorgenommen werden, um die zuvor gesteckten Ziele erfüllen zu können. Dazu gehörte neben der Suche nach Tags auch eine Möglichkeit, Suchanfragen speichern und abonnieren zu können. Die Lösungswege zu diesen Aufgaben werden im Folgenden erläutert.

11.1. Behebung des Speicherlecks

Nachdem in der Analyse ein Speicherleck gefunden worden war (siehe auch Kapitel 5 auf Seite 33), war es nun die Hauptaufgabe, dieses Leck zu beseitigen, um Speicherüberläufe durch Zugriffe auf die `persist`-Methode zu verhindern.

Änderungen im Quelltext Bei der Analyse der betroffenen Methode fiel auf, dass neben der durch die vorhergehende PG eingeführte Zeitmessung lediglich ein `saveOrUpdate()` auf das übergebene Objekt angewendet wurde, ohne zu gewährleisten, dass die Daten nach dem Schreiben in die Datenbank auch aus dem Speicher entfernt werden. Zu diesem Zweck bietet Hibernate die beiden Methoden `flush()` und `clear()` an. Die `flush`-Methode sorgt dafür, dass alle sich noch im Puffer befindlichen Daten in die Datenbank geschrieben werden. Durch den Aufruf der `clear`-Methode wird der Puffer komplett geleert und der Speicher wieder freigegeben, was dem Garbage-Collector ermöglicht, eventuell noch vorhandene, bereits persistierte Instanzen, freizugeben. In diesem Fall bietet es sich an erst einen Aufruf von `flush()` durchzuführen, um zu gewährleisten, dass keine Daten, die noch nicht persistiert wurden, gelöscht werden. Nach erfolgtem Schreiben der Pufferdaten in die Datenbank kann `clear()` aufgerufen werden. Listing 11.1 zeigt, wie die persist-Methode nach dem Hinzufügen der Methodenaufrufe für `flush()` und `clear()` aussieht.

```
[...]
Session session = getSession();
session.saveOrUpdate(entity);

//Edit to prevent hibernate memory leak
session.flush();
session.clear();
```

[. . .]

Listing 11.1: Auszug aus der persist-Methode

Ergebnis Nach den Änderungen an der `persist`-Methode wurden die bereits in der Analyse durchgeführten Tests wiederholt (siehe auch 5.2 auf Seite 34). Die Ergebnisse bestätigten die Vermutung, dass der Ursprung des Speicherlecks im Persistence-Manager lag. Nun war die massive Zunahme der Log-Eintrag-Instanzen nicht mehr zu beobachten.

Da die betroffene `persist`-Methode in vielen Teilen von oCOBS genutzt wird, liegt die Vermutung nahe, dass nun auch Speicherprobleme im Bereich des Imports gelöst sein sollten. Dies muss jedoch noch durch nachfolgende Tests bestätigt werden. Zudem ist nicht ausgeschlossen, dass in anderen Bereichen von oCOBS noch Speicherlecks anzutreffen sind, welche die Suchfunktion nicht tangieren und deshalb im Rahmen des Profilings nicht identifiziert werden konnten.

11.2. Weiterentwicklung der Suche

Durch den Einbau der Tagging-Funktion für angemeldete Benutzer (siehe hierzu auch 8.3 auf Seite 53) war es nötig, Änderungen an der Suchfunktion vorzunehmen. Dem Benutzer sollte es ermöglicht werden, über die Suchseite nach Tags suchen zu können. Außerdem sollte die Suche nach Tags angesteuert werden, sobald in einer Tag-Cloud ein Tag angeklickt wird. Für die Implementierung dieser Suche boten sich zwei Alternativen an. Eine Möglichkeit wird dargestellt durch die Erweiterung des existierenden Suchdialekts um die Suche nach Tags. Alternativ stand die Möglichkeit zur Wahl, eine separate Suche zu implementieren, die direkt in der Datenbank nach bestimmten Tags sucht. Aus mehreren Gründen fiel die Wahl auf die direkte Datenbankabfrage. Zum einen wird die Tagsuche sehr häufig verwendet und muss sich durch eine hohe Performanz auszeichnen. Dies lässt sich durch den Einsatz einfach gehaltener Datenbankabfragen am ehesten realisieren. Außerdem hätte die Anpassung des äußerst komplex implementierten Suchdialekts den Zeitrahmen der PG gesprengt. Nach dieser Entscheidung wurde im Zuge der Implementierung der Tagsuche die Suchmaske um eine Auswahlbox ergänzt, die es ermöglicht, neben Publikationen und Autoren auch eine Suche nach Tags auswählen zu können.

11.3. Abonnieren von Suchanfragen

Eines der Ziele unserer Projektgruppe war es, den oCOBS-Benutzern das Abonnieren ihrer Suchanfragen zu ermöglichen. Dies hat den Vorteil, dass diese sich über neue Publikationen zu einem Thema informieren können. Um dieses Ziel zu erreichen,

wurden verschiedene Ansätze in der Projektgruppe diskutiert und miteinander verglichen. Im Wesentlichem stand zur Diskussion, ob die gewünschte Funktionalität mit einer eigenen Lösung oder bereits etablierten Standards realisiert werden soll.

Für eine eigene Lösung sprach die Tatsache, dass hierbei am meisten Freiraum bei der Implementierung gegeben ist. Zugleich würde man aber an vielen Stellen bestehende Standards vernachlässigen. Denkbar wäre es gewesen, das Abonnieren von Suchanfragen ähnlich der vorhandenen Funktionalität des Merkzettels (siehe Kapitel 4.4 auf Seite 32) zu gestalten. Dies hätte aber zur Folge gehabt, dass die Benutzer manuell die oCOBS-Internetseite aufrufen müssten, um zu erfahren, ob für eine Suchanfrage bereits neue Publikationen vorliegen. Dies könnte dem Benutzer möglicherweise zu aufwendig sein und ihn von der Verwendung von oCOBS abhalten.

Da eines unserer Nebenziele war oCOBS möglichst einfach und damit attraktiv für neue Benutzer zu gestalten, entschloss sich die Projektgruppe dafür, bei bestehenden Standards zu bleiben. So fiel die Wahl auf die Technologie *RSS*, die sich seit mehr als zehn Jahren im Internet etabliert hat. Der folgende Abschnitt erklärt zusammenfassend diese Technologie.

11.3.1. RSS

Bei *RSS (Really Simple Syndication)* [49] handelt es sich um einen im Internet etablierten Standard für den Nachrichtenaustausch und das Mitteilen von Veränderungen an einer Internetseite. Dabei werden meistens auf den zugehörigen Internetseiten sogenannte *RSS-Channels* (häufig auch als *RSS-Feeds* bezeichnet) angeboten. Deren Inhalt ähnelt sehr häufig dem der eigentlichen Internetseite in gekürzter Form.

Bezogen werden können diese RSS-Feeds über einen *RSS-Reader*, eine Software, die in regelmäßigen Abständen nach Aktualisierungen sucht. Der RSS-Feed liegt hierbei als wohlgeformte XML-Datei vor, die der RSS-Reader nach Neuigkeiten durchsucht, und diese dem Benutzer visualisiert. Weiterhin wird die Darstellung von RSS-Feeds bereits von vielen der heute gängigen Internetbrowser unterstützt (siehe Abbildung 11.1 auf der nächsten Seite).

Bei oCOBS wurden auf diese Weise drei RSS-Channels implementiert. Zwei davon wurden dabei für die Suche nach Publikationen zur Verfügung gestellt. Einer davon dient als RSS-Feed für die allgemeine Suche und einer für die Suche nach Tags (siehe Abbildung 11.2 auf Seite 84). Ein dritter RSS-Channel erscheint auf den Seiten der Autoren und bietet so eine einfache Möglichkeit, sich über neue Publikationen dieser zu informieren.

Im Rahmen der Implementierung wurde ein Servlet (`RSSServlet`) erstellt. Dieses wertet eine übergebene Suchanfrage mithilfe der oCOBS-Suche aus, liefert das Ergebnis allerdings als XML-/RSS-Datei, anstatt der sonst üblichen HTML-Datei, zurück.

Dabei können dem Servlet zusätzlich zum Suchtext und der Suchart (Suche nach Publikationen oder Suche nach Tags) noch weitere Parameter übergeben werden.

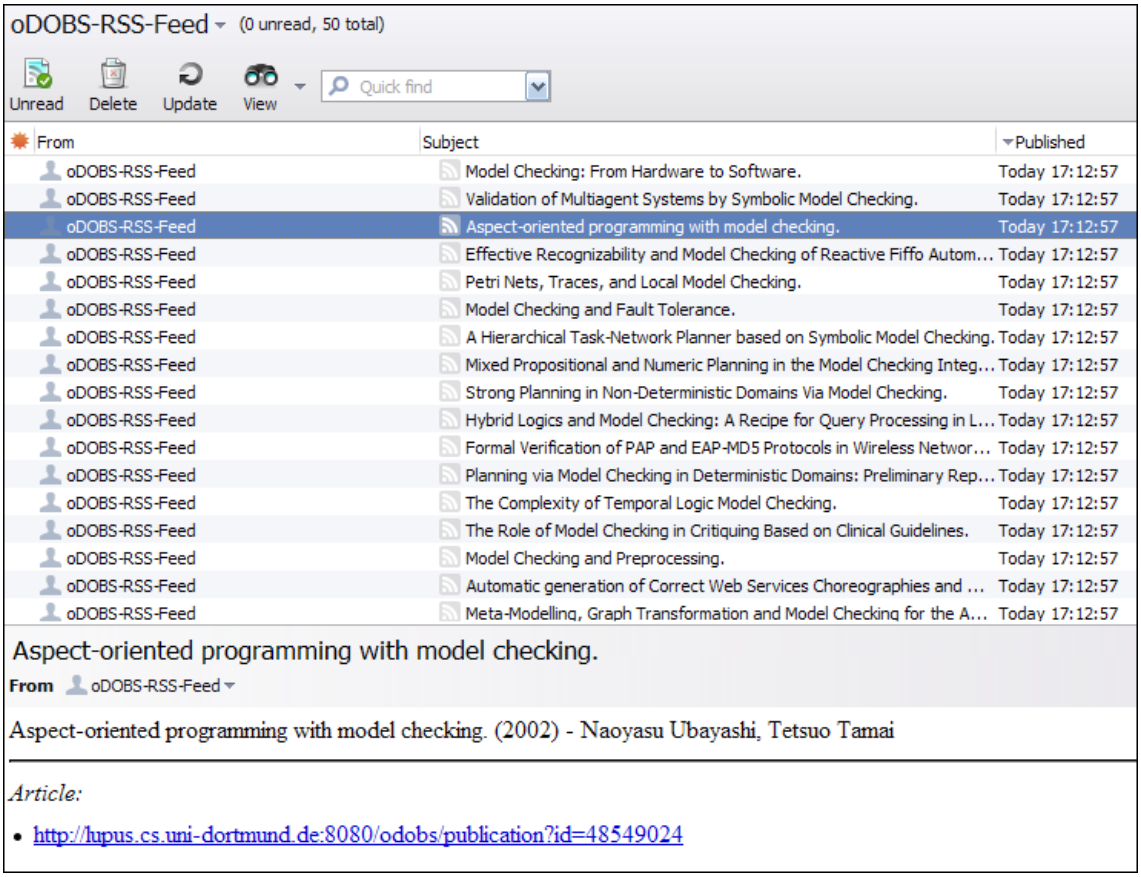

Abbildung 11.1.: RSS-Feed Darstellung in Opera [38]

So kann bestimmt werden nach welchem Kriterium (Titel, Jahr oder Ranking) die Ergebnisse wie (aufsteigend oder absteigend) sortiert werden sollen. Auf diese Weise können beim Anbieten eines RSS-Feeds auf einer Suchseite alle Kriterien der herkömmlichen Suche übernommen werden. Die Anzahl der Ergebnisse wird allerdings aus Gründen der Performance auf 100 reduziert.

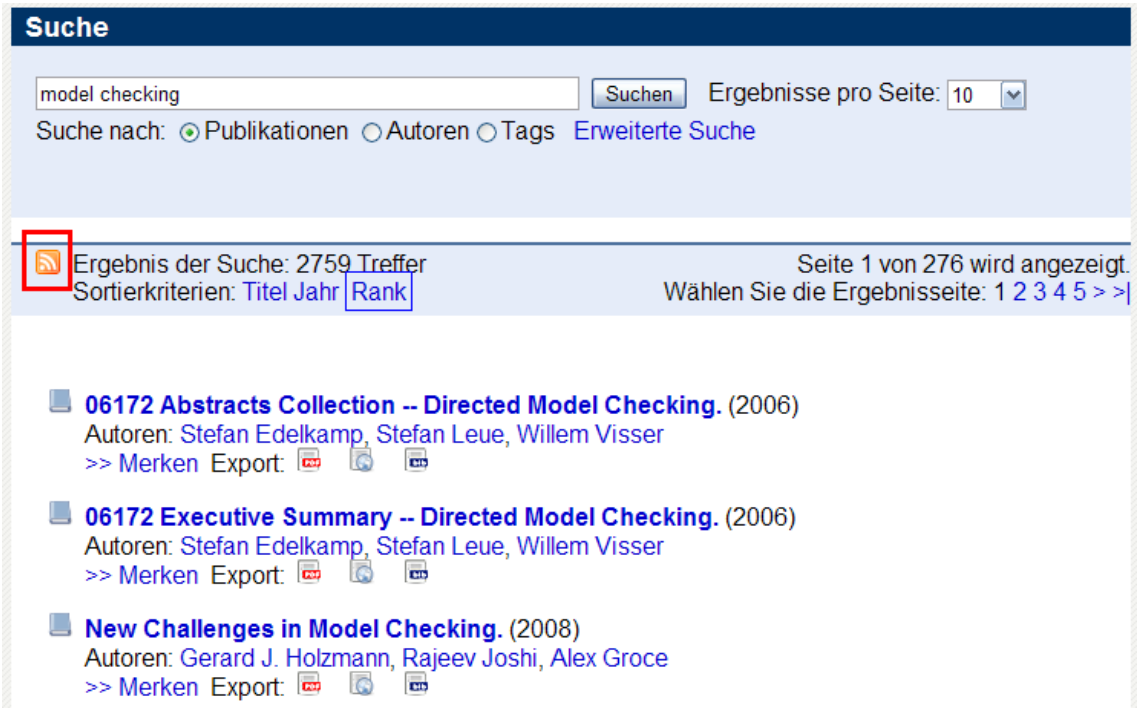

Abbildung 11.2.: Ergebnisseite mit RSS-Feed

12. Adminclient

12.1. Neue Funktionen des Admin-Clients

Zusätzlich zu denen in Kapitel 6 auf Seite 36 beschriebenen Funktionen wurden dem Admin-Client neue Funktionalitäten hinzugefügt, um den Administratoren Werkzeuge zu geben, um die neue Funktionalitäten des Web-Clients, wie die Erstellung der neuen oCOBS-Benutzer, Tags und Kommentaren, verwalten zu können. Die Administratoren können unerwünschte Beiträge löschen oder Nutzer sperren und löschen. Des Weiteren können Nutzer unangemessene Beiträge melden, welche dann den Administratoren als Aufgabe vorliegen.

All diese Funktionen befinden sich in einem neuen Reiter in der Adminoberfläche der `oCOBS Benutzerverwaltung`.

oCOBS Benutzer suchen

Dieser Reiter in der `oCOBS Benutzerverwaltung` ermöglicht dem Administrator sich alle oCOBS Benutzer anzeigen zu lassen, oder gezielt nach oCOBS Nutzern zu suchen. Diese können dann gesperrt oder komplett gelöscht werden. Sperren bedeutet, dass der Nutzer sich nicht mehr anmelden kann. Seine Daten und verfassten Kommentare, Tags und Bewertungen bleiben allerdings erhalten. Des Weiteren kann sich der Administrator alle vom ausgewählten Nutzer verfassten Tags und Kommentare anzeigen lassen und diese gegebenenfalls bearbeiten.

Tags suchen

Unter dem Reiter `Tags suchen` der `oCOBS Benutzerverwaltung` kann sich der Administrator sämtliche Tags anzeigen lassen oder alternativ gezielt nach Tags suchen. Diese werden zusammen mit ihrem Verfasser und der Publikation, an der sie getaggt wurden, aufgelistet. Der Administrator kann diese nun mit einem einfachen Klick entfernen.

Kommentare suchen

Der Reiter `Kommentare suchen` verhält sich analog zum Reiter `Tags suchen`. Es können wiederum alle Kommentare angezeigt werden oder gezielt nach diesen gesucht werden. Auch den Kommentaren werden ihre Verfasser und entsprechenden

Publikationen zugeordnet und Kommentare werden durch Betätigung des entsprechenden Löschfeldes entfernt.

Neue Aufgaben

Durch die Neuerungen im Web-Client sind zwei neue Tasks für Administratoren notwendig geworden, nämlich „Tag melden“ und „Kommentar melden“. Diese werden nach Aufruf im Web-Client dem Administrator im Aufgaben-Reiter der Adminoberfläche angezeigt und sind direkt mit der entsprechenden Suchfunktion in der oCOBS Benutzerverwaltung verknüpft, falls eine Bearbeitung nötig sein sollte (siehe auch Kapitel 6.1 auf Seite 38).

Teil IV.

Schlußbemerkungen

13. Zusammenfassung

Wie in Kapitel 1 auf Seite 2 beschrieben, ist oCOBS eine Weiterentwicklung des DBLPs der Universität Trier. Die aktuellen Erweiterungen setzen die Arbeit von zwei vorherigen Projektgruppen der Fakultät für Informatik an der Technischen Universität Dortmund fort. Nach einer gründlichen Einarbeitung in die oCOBS-Architektur wurde das erste Augenmerk auf die Behebung von Stabilitätsproblemen und dem Aufsetzen einer Live-Instanz gelegt, dies war für die weiteren Arbeiten unerlässlich.

Die weitere Arbeit unterteilte sich in drei Aufgabenbereiche. Die Arbeitsgruppe „Datenbank“ hatte die Aufgabe das Datenmodell entsprechend der Änderungen, die sich aus den neuen Funktionen ergeben, anzupassen. Die Arbeit beschränkte sich zum großen Teil auf Modifikationen, die mittels Hibernate durchgeführt wurden. Zudem hat sich diese Arbeitsgruppe mit dem Import der Daten aus dem DBLP beschäftigt (vgl. Kapitel 10 auf Seite 74). Hier wurde der Publikationsimport vollständig neu implementiert, was eine bessere Bedienbarkeit und höhere Performanz zur Folge hatte. Zusätzlich wurde mit der Arbeitsgruppe „Webclient“ zusammen an den neuen Benutzerfunktionen, speziell dem Tagging gearbeitet.

Die Entwickler dieses Teams haben zudem die Benutzerverwaltung zur Handhabung der oCOBS-Benutzer realisiert (siehe Kapitel 9.2 auf Seite 70). Basierend darauf wurde die Anreicherung der Publikationen um Metadaten realisiert. Hierzu gehörte das Tagging, sowie die Bewertungs- und Kommentarfunktionen (vgl. Kapitel 8 auf Seite 50). Außerdem wurde von dieser Gruppe das Abonnieren von Suchanfragen mittels RSS realisiert (siehe Kapitel 11.3 auf Seite 81).

Die dritte Arbeistgruppe befasste sich mit den Änderungen am „Adminclient“ (siehe Kapitel 12 auf Seite 85). Dazu gehört speziell die Administration der neuen Benutzerfunktionen, wie z.B. dem administrativen Verwalten der Tags und Kommentare. Zusätzlich wurde eine Verwaltung der Benutzer eingerichtet, hier können oCOBS-Nutzer geändert, gesperrt oder gelöscht werden. Die Gruppe befasste sich zudem mit den Webservices, die angepasst und erweitert werden mussten, um die Kommunikation zwischen Webclient-Backend und Adminclient zu ermöglichen.

Gemeinsam wurde zudem an der Realisierung einer neuen Navigationsform über Tag-Clouds gearbeitet, die es ermöglicht neue relevante Publikationen oder Benutzer mit ähnlichen Interessengebieten über Tags zu finden (siehe Kapitel 8.3.3 auf Seite 61). Des Weiteren wurde die Suche dahingehend erweitert, dass jetzt auch nach Tags gesucht werden kann (Kapitel 11.2 auf Seite 81).

14. Ausblick

Dieses abschließende Kapitel soll kurz aufzeigen, was an oCOBS noch verbessert werden kann, bzw. welche Möglichkeiten existieren, um das System, aber auch die Benutzerfreundlichkeit zu optimieren.

Grundsätzlich besteht die Möglichkeit, die in oCOBS verwendete Software auf eine aktuellere Version zu bringen, da der Support für einige ältere Versionen von Herstellerseite eingestellt wurde. Dies betrifft vor allem den Applikationsserver JBoss, der bei oCOBS in Version 4.0.5 verwendet wird, aber schon in Version 6 zur Verfügung steht. Weiterhin existiert auch ein komplett überarbeitetes Maven 2, welches sich sehr von der bei oCOBS verwendeten Version 1.0.2 unterscheidet. Des Weiteren können verwendete Bibliotheken auf neuere Versionen überprüft und gegebenenfalls auf diesen Stand gebracht werden. Neben Aktualisierungsmöglichkeiten an der Infrastruktur bieten sich auch Verbesserungen an den jeweiligen Komponenten von oCOBS an, die im Folgenden vorgestellt werden.

Datenbank

Eine Aufgabe, die sich an die Projektgruppe anschließt, ist eine engere Verzahnung der verschiedenen Subprogramme von oCOBS. Es wäre möglich die Dublettensuche mit dem Import zu verbinden, so dass beim Importvorgang schon mögliche Dubletten gefiltert werden. Außerdem sollten die Dublettensuche und vor allem der Import besser in die Adminoberfläche integriert werden, so dass der Importvorgang aus dieser gestartet werden kann und Statusmeldungen eingesehen werden können. Eine weitere Aufgabe könnte die Verbesserung der Bedienbarkeit und Installation von oCOBS sein. Diese kann z.B. durch ein automatisiertes Skript oder eine Installationsroutine vorgenommen werden. Das Vor- und Nachbereiten der Datenbank, inklusive des Erstellens von Indizes etc., kann dadurch ebenfalls optimiert und automatisiert werden.

Webclient

Der Webclient wurde durch eine integrierte Benutzerverwaltung und die Möglichkeit Kommentare und Bewertungen zu Publikationen hinzuzufügen deutlich attraktiver für neue Benutzer gestaltet. Jedoch bietet auch der aktuelle Status noch viel Raum für weitere Verbesserungen und Anpassungen.

Die neuen Funktionen wurden teilweise unter Verwendung aktueller Web-Technologien (z.B. AJAX) in oCOBS integriert. Für die zukünftige Entwicklung von oCOBS wäre es demnach wünschenswert, wenn die alten Funktionen, die von den beiden vorherigen PGs implementiert wurden, ebenfalls um neue Technologien ergänzt würden, um einen konsistenteren Eindruck beim Benutzer zu hinterlassen. Beispielsweise könnten die Suchergebnisse ebenfalls mit AJAX angezeigt werden. Auch das Ein- und Ausblenden von Informationen jeweiliger Publikationen auf den Publikationsseiten könnte zur optischen Verbesserung mit Hilfe von JavaScript umgesetzt werden.

Eine Erweiterung der Benutzerfunktionen könnte die Attraktivität von oCOBS weiter erhöhen. Je mehr Möglichkeiten dem Benutzer gegeben werden, die Seite inhaltlich mitzugestalten, desto eher lässt sich annehmen, dass er oCOBS weiter nutzen wird. Innerhalb dieser Überlegung stellt sich dann die Frage, was eine solche Erweiterung für Funktionen beinhalten könnte. Im Folgenden werden einige theoretische Funktionserweiterungen vorgestellt:

- Integration eines sozialen Netzwerkes, in dem Benutzer die Möglichkeit erhalten sich mit anderen Benutzern über Publikationen mittels eines Nachrichtensystems auszutauschen.
- Erstellen von Gruppen zu bestimmten Fachthemen, in die Benutzer eingeladen werden können, um gemeinsam zu diskutieren und zu arbeiten. Hier wäre auch das Verwalten einer gemeinsamen Publikationsmenge denkbar.
- Umfangreichere Anpassung des Benutzerprofils. Bisher kann der Benutzer in seinem Profil nur wenige Daten hinterlegen. Dies könnte für eine weitere Personalisierung der oCOBS-Seite sorgen, die den Benutzer mehr an oCOBS bindet. Dazu müsste zunächst ermittelt werden wie das derzeitige Benutzerprofil genutzt wird und ob Bedarf nach Erweiterungen seitens der Benutzer besteht.
- Aktualisieren des Designs von oCOBS. Dies könnte moderner und schöner gestaltet werden.

Adminclient

Erweiterungen des Adminclients werden in der Zukunft mit Weiterentwicklungen des Webclients einhergehen, um äquivalente Methoden für die Korrektur von benutzergenerierten Inhalten zu schaffen. Des Weiteren könnte überlegt werden, ob es sinnvoll wäre, das Rollensystem, welches im Adminclient existiert, auch auf die Nutzer des Webclients auszudehnen. So könnten ausgewählte Nutzer den Datenbestand pflegen und Moderationsfunktion einnehmen. Wie bereits erwähnt wäre es sinnvoll, den Import in den Adminclient einzugliedern und eine Historie über bereits erfolgte Imports anzulegen.

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] Mitja Bamberger. *Seminarausarbeitung: Java Server Faces*, 2009.

[2] Hans Bergsten. *Java Server Pages*. O'Reilly, 2001.

[3] Projektgruppe 514 DoPAC: Dortmund Online Public Access Catalog. *oDOBS Wiki der PG 514*. `http://odobs.cs.uni-dortmund.de/wiki`.

[4] Projektgruppe 514 DoPAC: Dortmund Online Public Access Catalog. *Endbericht der PG 514 DoPAC*, Juni 2008.

[5] Jim Farley and William Crawford. *Java Enterprise in a Nutshell*. O'Reilly, 3rd edition, 2005.

[6] Johannes Henkel. *Java Bibtex API*. `http://www-plan.cs.colorado.edu/henkel/stuff/javabib/20040801/javadoc/`.

[7] Oliver Ihns, Dierk Harbeck, Stefan Heldt, and Holger Koschek. *EJB 3 professionell*. dpunkt.verlag, 1. edition, 2007.

[8] Projektgruppe 494 L2EE: Lightweight Process Coordination & J2EE. *Endbericht der PG 494 L2EE*, März 2007.

[9] Arne Koschel, Stefan Fischer, and Gerhard Wagner. *J2EE/Java EE kompakt*. Elsevier, 2. edition, 2006.

[10] Jakob Langer. *Seminarausarbeitung: oDOBS Adminclient und Backend*, 2009.

[11] Michael Ley. *DBLP - Some Lessons Learned*. *PVLDB*, 2(2):1493–1500, 2009.

[12] Benedikt Mättig. *Seminarausarbeitung: oDOBS - Datenmodell und Import*, 2009.

[13] Projektgruppe 540 Semantic oDOBS. *oDOBS Wiki der PG 540*. `http://pavo.cs.uni-dortmund.de/mediawiki`.

[14] Michael p. Papazoglu. *Principles and Foundations of Web Services: An Holistic View (Technologies, Business Drivers, Models, Architectures and Standarts)*. Prentice Hall, 2007.

[15] Gunther Popp. *Konfigurationsmanagement mit Subversion, Ant und Maven.* dpunkt.verlag GmbH Heidelberg, 2006.

[16] Christoph Schlüter. *Seminarausarbeitung: Vergleich von Konkurrenzsystemen für oDOBS*, 2009.

[17] Ralph Steyer. *AJAX Frameworks.* Addison-Wesley Verlag, 2008.

[18] Vincent Massol und Timothy M. O'Brien. *Maven. Das Entwicklerheft.* O'Reilly Verlag, 2005.

[19] Christian Winter. *Seminarausarbeitung: Projektmanagement: Maven und Subversion*, 2009.

[20] Yu Xue. *Seminarausarbeitung: Resource Description Framework*, 2009.

Internetquellen

[21] DBLP - Digital Bibliography & Library Project. http://www.informatik.uni-trier.de/~ley/db/. zuletzt besucht am 25.4.2010.

[22] Java Application Building Center. http://jabc.cs.uni-dortmund.de/. zuletzt besucht am 11.02.2010.

[23] Java Enterprise Edition - Projektseite. http://www.oracle.com/technetwork/java/javaee/. zuletzt besucht am 15.11.2010.

[24] Java Standard Edition - Projektseite. http://www.oracle.com/technetwork/java/javase/. zuletzt besucht am 15.11.2010.

[25] JBoss Profiler Homepage. http://www.jboss.org/jbossprofiler/. zuletzt besucht am 26.4.2010.

[26] JCaptcha Homepage. http://jcaptcha.sourceforge.net/. zuletzt besucht am 26.4.2010.

[27] Jena Homepage. http://jena.sourceforge.net/. zuletzt besucht am 25.4.2010.

[28] JOID Homepage. http://code.google.com/p/joid/. zuletzt besucht am 25.4.2010.

[29] JOpenID Homepage. http://code.google.com/p/jopenid/. zuletzt besucht am 25.4.2010.

[30] JSON Homepage. http://json.org/. zuletzt besucht am 8.7.2010.

[31] myOpenID Homepage. https://www.myopenid.com/. zuletzt besucht am 26.4.2010.

[32] oDOBS - Dortmunder Online-Bibliographieservice. www.odobs.de. zuletzt besucht am 25.10.2010.

[33] Odobs Entwickler-Wiki. `http://lupus.cs.uni-dortmund.de/wiki/index.php`. zuletzt besucht am 25.4.2010.

[34] OpenID Attribute Exchange. `http://openid.net/specs/openid-attribute-exchange-1_0-05.html`. zuletzt besucht am 25.4.2010.

[35] OpenID Homepage. `http://openid.net/`. zuletzt besucht am 26.4.2010.

[36] OpenID UI Extension. `http://wiki.openid.net/f/openid_ui_extension_draft01.html`. zuletzt besucht am 25.4.2010.

[37] OpenID4Java Homepage. `http://code.google.com/p/openid4java/`. zuletzt besucht am 25.4.2010.

[38] Opera Browser. `www.opera.com`. zuletzt besucht am 25.10.2010.

[39] Sesame Homepage. `http://www.openrdf.org/`. zuletzt besucht am 25.4.2010.

[40] *delicious.com*. `http://delicious.com/`. zuletzt besucht am 25.4.2010.

[41] *Flickr Tag-Cloud*. `http://www.flickr.com/photos/tags/`. zuletzt besucht am 25.4.2010.

[42] *Hibernate*. `https://www.hibernate.org/`. zuletzt besucht am 25.4.2010.

[43] *Java Server Faces*. `http://java.sun.com/javaee/javaserverfaces/`. zuletzt besucht am 25.4.2010.

[44] World Wide Web Consortium. `www.w3.org`. zuletzt besucht am 10.09.2010.

[45] PG 540. JBoss Migration. `http://pavo.cs.uni-dortmund.de/mediawiki/index.php/JBoss_Migration`, 2010. zuletzt besucht am 30.7.2010.

[46] Nadia Alramli. jQuery Confirm Plugin. `http://nadiana.com/jquery-confirm-plugin`. zuletzt besucht am 11.7.2010.

[47] Apache. Maven. `http://maven.apache.org/`. zuletzt besucht am 7.7.2010.

[48] Apache. Subversion. `http://subversion.apache.org/`. zuletzt besucht am 7.7.2010.

[49] RSS Advisory Board. Rss 2.0 specification. `http://www.rssboard.org/rss-specification`. zuletzt besucht am 9.7.2010.

[50] CiteSeer^{X}. CiteSeer^{X} Homepage. `http://citeseerx.ist.psu.edu/`. zuletzt besucht am 26.4.2010.

[51] CiteULike. CiteULike Homepage. `http://www.citeulike.org/`. zuletzt besucht am 26.4.2010.

[52] codehaus.org. *Woodstox - High-performance XML processor*. `http://woodstox.codehaus.org/`. zuletzt besucht am 25.4.2010.

[53] Roger Kitain Ed Burns. Jsr 314: Java Server Faces 2.0. `http://www.jcp.org/en/jsr/detail?id=314`, 2009. zuletzt besucht am 19.7.2010.

[54] Steve Friedl. SQL Injection Attacks by Example. `http://unixwiz.net/techtips/sql-injection.html`. zuletzt besucht am 25.4.2010.

[55] Google. Google Schoolar Homepage. `http://scholar.google.de/`. zuletzt besucht am 01.11.2010.

[56] Red Hat. JBoss Aplication Server. `http://www.jboss.org/jbossas`. zuletzt besucht am 7.7.2010.

[57] Sun Microsystems. Java EE 5 Tutorial. `http://java.sun.com/javaee/5/docs/tutorial/doc`. zuletzt besucht am 5.7.2010.

[58] Oren Patashnik. *Bibtexing*. `http://amath.colorado.edu/documentation/LaTeX/reference/faq/bibtex.pdf`, 1988. zuletzt besucht am 26.4.2010.

[59] DBLP Digital Bibliography & Library Project. *Dateien für den DBLP-Import*. `http://dblp.uni-trier.de/xml/`. zuletzt besucht am 25.4.2010.

[60] John Resig. jQuery Homepage. `http://jquery.com/`. zuletzt besucht am 7.7.2010.

[61] J. Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. `http://tools.ietf.org/html/rfc4511`. zuletzt besucht am 8.7.2010.

[62] Sun. Javadoc Homepage. `http://java.sun.com/j2se/javadoc/`. zuletzt besucht am 25.4.2010.

[63] wikipedia.org. Wikipedia - Die freie Enzyklopädie. `http://de.wikipedia.org/`. zuletzt besucht am 25.4.2010.

[64] David Yu. dyuproject Homepage. `http://code.google.com/p/dyuproject/`. zuletzt besucht am 25.4.2010.