Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektausarbeitung

Raoul Pascal Pein

Retrieval System "Golden Retriever"

Fakultät Technik und Informatik
Studiendepartment Informatik

Thema der Projektausarbeitung
Retrieval System "Golden Retriever"

Stichworte
Retrieval, Content Based Image Retrieval, Bildsuche, Collaborative Workspace

Kurzzusammenfassung
Diese Arbeit beschreibt das Retrieval System "Golden Retriever", das im Verlauf eines einsemestrigen Projektes an der HAW Hamburg entstanden ist. Es ist die Fortführung eines vorherigen CBIR-Projektes, das in eine Collaborative Workspace Umgebung eingebettet wurde. Dazu wurde der vorhandene Code überarbeitet und an die besonderen Gegebenheiten des Labors angepasst.
Der resultierende Prototyp kann Bilder mit verschiedenen Techniken auf Ähnlichkeit überprüfen und mit relativ geringem Mehraufwand auch andere Dateitypen vergleichen. Die meisten Funktionen lassen sich über Servlets aufrufen.

Inhaltsverzeichnis

1 Einleitung

Das im Folgenden beschriebene Projekt ist Teil des Gemeinschaftsprojektes „Collaborative Workspace“ im Rahmen des Masterstudienganges Informatik der HAW Hamburg.

Ziel war die Implementation eines Suchdienstes, der die besonderen Gegebenheiten des sich im Aufbau befindlichen CW-Labors bestmöglich unterstützt.

Der Ausgangspunkt ist eine vorherige Bachelor-/Diplomarbeit (10). In dieser wurde ein Prototyp für einen Content Based Image Retrieval Dienst implementiert. Auf dieser Basis wurde ein neuer Prototyp erstellt, der Schnittstellen zur gemeinsamen Nutzung im Labor bietet.

2 Konzept

Dieser Abschnitt stellt die ursprünglichen Ziele und Pläne des Projektes dar. Diese beruhen auf den Besprechungen des Teams in der Anfangsphase.

2.1 Projektziele

Der geplante Suchdienst basiert auf einem vorher angefertigten CBIR-Prototypen. Dieser sollte um eine einfach zu nutzende Schnittstelle erweitert werden, die möglichst flexibel genutzt werden kann. Weitere Ziele waren die Verbesserung von Suchqualität, Geschwindigkeit und einer allgemeinen Überarbeitung des Codes. Außerdem soll die Unterstützung von anderen Datentypen als Bildern ermöglicht werden.

Die serverseitigen Teile des Retrievalsystems sollen auf Blade-Server von IBM laufen. Ziel ist es, diese „Intelligenz“ komplett aus dem Blickfeld der Anwender verschwinden zu lassen. Es soll nur noch die für die Interaktion notwendige Hardware im Arbeitsumfeld vorhanden sein.

Konkret wurde geplant, das Retrievalsystem statt auf einer Datenbank auf einem verteilten Dateisystem laufen zu lassen. Zu diesem Zweck soll die Cluster-Lösung „Lustre“ zum Einsatz kommen, das von Mykhaylo Kabalkin im Labor in Betrieb genommen wird.

Auf der Anwenderseite steht ein Projekt von Philipp Roßberger, das den Suchdienst nutzt. Hier wird die gemeinsame Verwaltung von Bildern im Labor unterstützt. Als Interaktionsmedien dienen zu diesem Zweck große, zum Teil touchfähige Bildschirme.

Zur Betonung der Metapher des „intelligenten Raums“ wurde weiterhin geplant, neu hinzu kommende Informationen mit zu berücksichtigen. Das bedeutet, sobald eine weitere Person den Raum betritt, werden automatisch die mitgebrachten Dateien analysiert und in das Gesamtsystem eingebunden. Passen beispielsweise neue Bilder direkt zu den vorhandenen Bildern auf den Displays, werden sie sofort mit angezeigt.

Eine weitere Idee war das Konzept der „Virtuellen Ordner“. Diese sollen nach Vorbild von iTunes (6) die Definition einer Suchanfrage enthalten. Auf Basis dieser Anfrage wird der Ordner dann mit Referenzen auf alle passenden Dateien gefüllt, so dass es für den Anwender so aussieht, als wären diese Dateien alle tasächlich im Verzeichnis enthalten. Eine dynamische Anpassung des Inhaltes auf Veränderungen zur Laufzeit kommt dabei der Idee des vollständig vernetzten Datenbestandes nahe.

Eine Zusammenarbeit mit dem „Rescue“-Projekt wurde ebenfalls angedacht. Hier könnten eine standortbasierte Suche beispielsweise Sensoren oder Personen in einem bestimmten Zielgebiet finden. Weiterhin ist eine Suche nach Sensoren mit kritischen Meßwerten denkbar.

2.2 Architektur

In diesem Abschnitt wird die Architektur auf zwei verschiedenen Ebenen beschrieben. Zum einen wird die globale Ebene beschrieben, in der die Schnittstellen zu den Partnerprojekten im Labor erkennbar sind. Zum anderen wird detaillierter auf die lokalen Gegebenheiten des Projektes eingegangen.

2.2.1 Global

Im Labor arbeiten zwei andere Projekte eng mit dem hier beschriebenen zusammen. Die in Abb. 1 gelb hinterlegten Kästen stellen jeweils ein paralleles Projekt dar.

Alternativ zu der lokalen Datenbank wird die Persistenzschicht von Mykhaylo Kabalkin realisiert. Diese bindet die Dateiysteme der Blade-Server, des NAS und beliebiger anderer Datenträger im Labor ein. Eine konkrete Schnittstelle wurde bisher noch nicht vereinbart. Im einfachsten Fall wird das System transparent wie ein normales Dateisystem verwendet.

Die Präsentationsschicht wird im Labor von Philipp Roßberger realisiert. Diese greift über Websservices auf den Suchdienst auf dem Server zu. Die Schnittstelle muß hauptsächlich die Suche nach verschiedenen Parametern und den Zugriff auf die verwalteten Bilder bieten.

2.2.2 Lokal

Das als Grundlage dienende System bietet bereits eine Java-Swing GUI und die Anbindung an eine MySQL-Datenbank. Dabei ist die GUI über Java-RMI vom serverseitigen Dienst entkoppelt. Das Ausgangssystem besitzt daher eine klassische 3-Schichten-Architektur mit Persistenz, Server und einem Fat-Client.

Aufbauend auf dem alten Retrievalsystem, das aus *Retrieval Client*, *Retrieval Server*, *Persistence* und *Administration* besteht, werden weitere Komponenten ergänzt (Abb. 2).

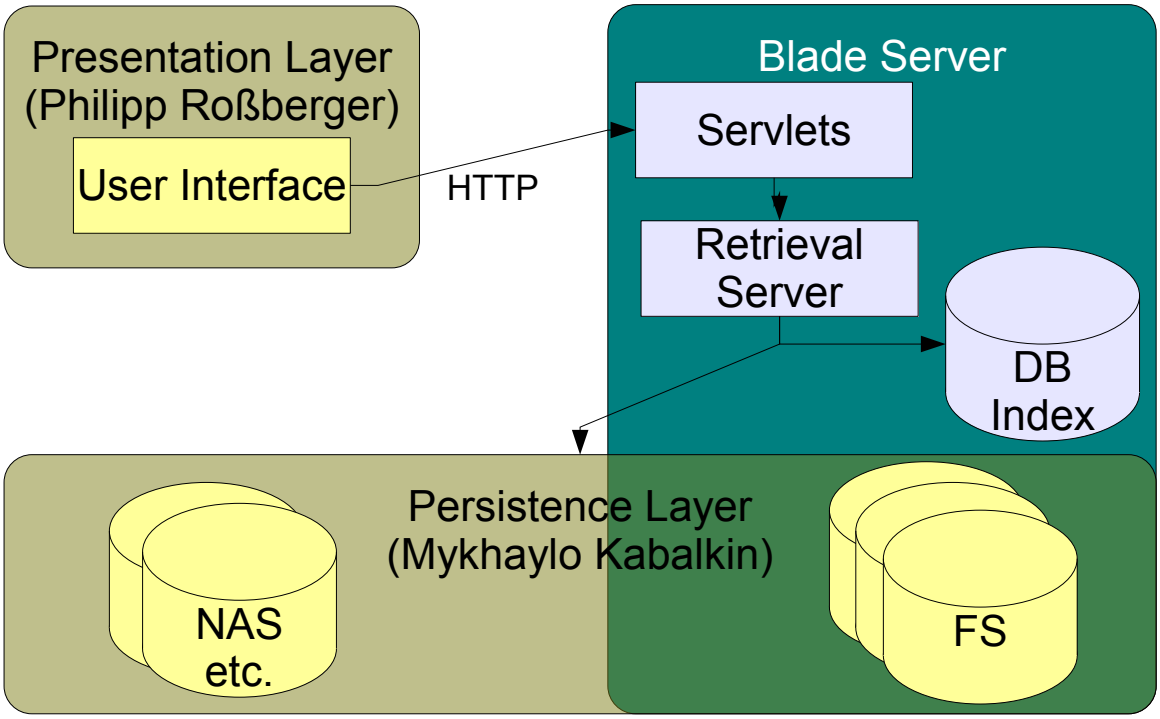

Abbildung 1: Einbindung in die Laborarchitektur. Hellblau hinterlegte Komponenten sind Teil des Prototypen. Gelbe Komponenten werden in anderen Projekten behandelt.

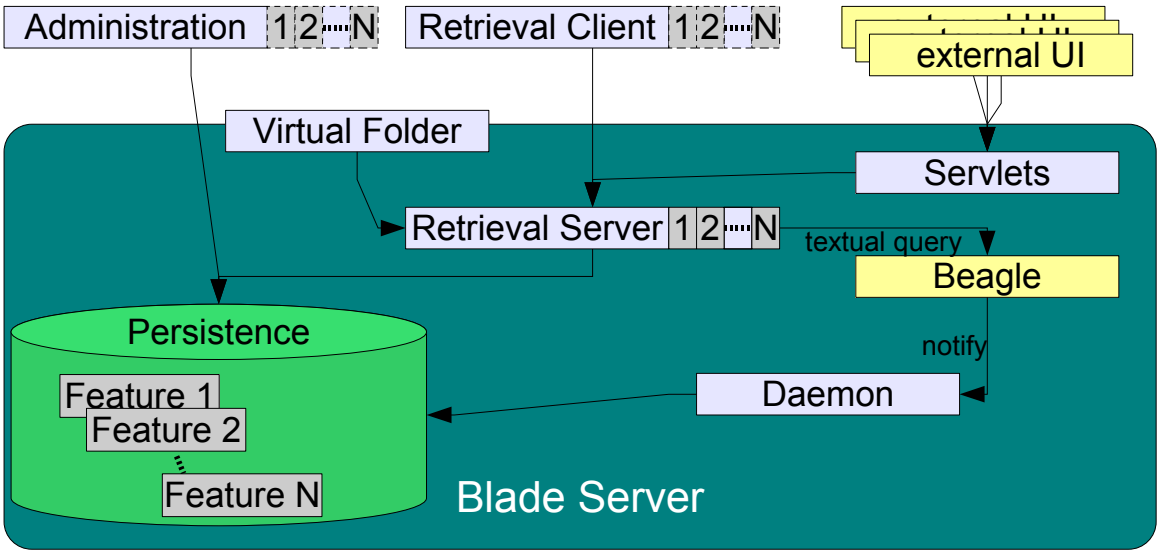

Abbildung 2: Die Architektur des Prototypen. Hellblau hinterlegte Komponenten sind Teil des Prototypen. Gelbe Komponenten sind extern angebunden. Die grüne Persistenz baut optional auf einem verteilten Dateisystem auf.

Kernstück bleibt weiterhin der *Server*, der die eigentliche Suche durchführt. Dazu können verschiedene Module integriert werden, die jeweils einen Aspekt bei der Suche abbilden, wie zum Beispiel Histogramme oder Schlagworte.

Der alte *Client* auf RMI-Basis bleibt ebenfalls bestehen. Optional können hier Panels in die GUI integriert werden, die eine aspektspezifische Eingabe der Suchanfrage ermöglichen. Neu an dieser Stelle sind die *Servlets*, die ebenfalls auf den Server zugreifen. Diese setzen externe HTTP-Requests so um, dass der Server sie verarbeiten kann. Die Ergebnisse werden größtenteils als XML zurückgegeben.

Über die *Administration* können neue Dateien in den Suchindex aufgenommen werden. Diese manuell gesteuerte Funktionalität soll durch einen Hintergrundprozess (*Daemon*) ergänzt werden, der zur Laufzeit neue Daten erkennt und automatisch indexiert.

Der Server bekommt zudem die Möglichkeit, externe Suchdienste zu verwenden. Für eine textbasierte Suche bietet sich *Beagle* (12) an, das eine Liste von Dateien auf Grund einer Suchanfrage generieren kann. Mit Hilfe dieser Liste kann dann der eigene Suchraum beschränkt werden. Ferner verwendet Beagle einen Live Indexing Mechanismus, der auf inotify aufbaut. Dieser kann verwendet werden, um dem Daemon neue oder veränderte Dateien direkt zu melden.

Die letzte zu implementierende Komponente ist der *Virtual Folder*. Dieser arbeitet auf Ebene des Dateisystems und nutzt ebenfalls die Suchfunktionen des Servers.

Alle Komponenten, die keine direkte Interaktion mit dem Benutzer erfordern, müssen auf dem Blade lauffähig sein. Einzig Administration, RMI-Client und weitere Benutzungsschnittstellen können auf einem beliebigen Gerät im Labor verwendet werden.

3 Laboraufbau

Im Folgenden werden Details des Laboraufbaus beschrieben. Da das CW-Labor parallel zum eigentlichen Projekt komplett aufgebaut werden muss, waren hier einige grundlegende Arbeiten notwendig. Technische Details zu der Installation einiger Komponenten werden im Anhang näher beschrieben.

3.1 Betriebssystem

Die Frage nach dem zu verwendenden Betriebssystem war relativ einfach zu beantworten. Das Vorgängerprojekt wurde komplett in Java geschrieben und ist somit auf allen gängigen Betriebssystemen lauffähig. Da der Suchdienst direkt über Webservices angesteuert werden kann, war hier die Wahl des Betreibssystems nebensächlich.

Einschränkungen in der Systemauswahl werden erst duch die Entscheidungen bezüglich des verwendeten Dateisystems relevant. Die Nutzung von Lustre legt die Verwendung von

Linux nahe, macht diese aber nicht zwingend erforderlich. Entscheidend für die Wahl von Linux waren das von Haus aus flexible Dateisystem und der Plan, Beagle einzusezen.

Die Distribution *SuSE Linux Enterprise Server 10.1* (13) ist als 64Bit Version für den Servereinsatz erhältlich. Ferner ist dort Beagle bereits vollständig in das Fenstersystem Gnome integriert. Die ältere Version SLES 9 ließ sich auf den Blade-Servern nicht installieren. Diese sollte zuerst eingesetzt werden, da Lustre offiziell noch nicht auf SuSE 10 getestet wurde.

3.2 Hardware

Die verwendete Hardware reicht von normalen PCs, über eine Netzwerkfestplatte (NAS) bis hin zu einem IBM-Bladecenter mit meheren Servern.

Die Entwicklung des Prototypen wurde parallel auf einem Windows-PC und einem Linux-PC (SLES 10.1) durchgeführt. Einige spezielle Tests konnten lediglich mit dem Linux-Gerät ausgeführt werden. Hierzu zählen Experimente mit Beagle, sowie dateisystemspezifische Dinge wie die Erstellung von hard/symbolic links.

Die Netzwerkfestplatte besteht aus 4 baugleichen 250 GB Festplatten, die über ein Raid-5 zusammengeschaltet sind. Eine der Platten dient als Backup, so dass für die Nutzung 750 GB zur Verfügung stehen. Auf diesem System werden die zu verwaltenden Bilder hauptsächlich gespeichert. Eine 1GBit-Anbindung an das Netzwerk sorgt für einen schnellen Zugriff auf die Daten.

Den eigentliche Kern des Retrievaldienstes bilden die Blades. Auf mindestens einem dieser Geräte wird der Dienst letzten Endes laufen. Da die eigentliche Entwicklung auf einem SLES 10.1 PC stattfindet, sind keine größeren Probleme bei der Portierung auf den SLES 10.1 Blade zu erwarten.

3.3 Externe Komponenten

Für die Realisierung des Projektes wurden bereits vorhandene Suchdienste auf ihre Tauglichkeit überprüft. Alle CBIR-basierten Systeme waren dabei entweder nicht verfügbar oder auf einem ähnlichen Stand, wie das eigene Basisprojekt. An textbasierten Suchdiensten waren hauptsächlich Goolge Deskop Search (4) und Beagle (12) in Verbindung mit Lucene interessant. Leider lassen sich beide Dienste nicht ohne weiteres um eine CBIR-Funktionalität erweitern. Stattdessen soll Beagle verwendet werden, um den textbasierten Teil der Suche zu übernehmen, da dieser bereits sehr performant ist.

Als Webserver und Datenbank wurde das Paket XAMPP (14) ausgewählt. Dieses stellt mit minimalem Konfigurationsaufwand einen Apache Webserver und eine MySQL Datenbank zur Verfügung. Mit diesen Diensten lassen sich die meisten notwendigen Funktionen testen. Zusätzlich ist Tomcat (1) notwendig, um die Servlets veraarbeiten zu können.

3.4 Werkzeuge

Für die Entwicklung kommen zwei verschiedene Werkzeuge zum Einsatz. Der Javacode wird mit Eclipse (3) erstellt, das zudem ein Tomcat-Plugin benötigt. Für die Arbeit am Beagle-Quellcode wird die IDE MonoDevelop (9) genutzt.

4 Realisierung

4.1 Implementation

4.1.1 Äußere Schnittstellen

Bei der Auswahl der Schnittstellen zu den anderen Teilprojekten standen mehrere Technologien zur Auswahl. Es wurden der Einsatz von Middleware wie RMI (11), CORBA (2) oder ICE (5), sowie die Verwendung leichtgewichtiger Webservices geprüft.

RMI Für den Einsatz von Java-RMI spricht, dass dieses bereits vom bestehenden Prototypen unterstützt wird. Die Kommunikation zwischen RetrievalClient und RetrievalServer findet vollständig über RMI statt. Der Nachteil ist die Festlegung auf die Sprache Java. Daher wird diese Möglichkeit vorerst nicht mehr weiter verfolgt.

CORBA/ICE CORBA und ICE sind sprachunabhängige Alternativen für Middleware. Der Einsatz einer dieser Technologien wird immer noch für die Nutzung zwischen Retrieval und Persistenz erwogen. Für die Nutzung der Retrievaldienste selbst ist diese Lösung zu schwergewichtig, da auch kleinere Geräte wie PDAs auf diesen Dienst zugreifen sollen.

Webservices Die Entscheidung für die Nutzung von Webservices ermöglicht es theoretisch jedem internetfähigen Gerät, den Retrievaldienst aufzrufen. Die Erstellung dynamisch generierter Webseiten wird dadurch ebenfalls vereinfacht. Wie bereits in Abschnitt 3.4 angedeutet, werden Servlets unter Verwendung von Tomcat genutzt.

4.1.2 Interne Schnittstellen

Innerhalb des Prototypen wird fast ausschließlich Java verwendet. Eine Ausnahme bilden die SQL-Zugriffe auf die Datenbank und die Teile, die mit Beagle direkt zusammenhängen (siehe 4.1.4).

4.1.3 Aspektmodule

Bisher wurden lediglich eine einfache Keywordsuche und 2 verschiedene Varianten von Histogrammen für die Ähnlichkeitssuche unterstützt. Zwei neue Aspekte wurden im Verlauf des Projektes hinzugefügt:

Kategorien Auf der semantischen Ebene kann die Einteilung der Bilder in Kategorien helfen, den Bestand zu sortieren und zu filtern. Für den Prototypen wurde ein hierarchischer Kategorienbaum realisiert. Jedes Bild kann beliebig vielen Knoten des Baumes zugeordnet werden. Dabei besitzt jeder Knoten eine eindeutige ID und einen Parent. Die Zuweisung der Kategorien muss von Hand für einzelne Bilder oder Bildergruppen vorgenommen werden. Die Suche wird über den Vergleich der IDs in Anfrage und jedem Datensatz geführt. Dieser Aspekt kann auf Grund seiner Eindeutigkeit später auch als Filterkriterium genutzt werden.

Wavelets Der zweite Aspekt befasst sich mit einer Waveletzerlegung von Bildern und ist somit eine unscharfe Suche, die den Bildinhalt vollautomatisch analysiert (7). Um diesen Algorithmus anwenden zu können, muss das Bild allerdings eine Seitenlänge von 2^N aufweisen. Eine Normierung der Eingangsbilder ist daher zwingend erforderlich.

4.1.4 Beagle

Beagle nutzt eine C#-Variante von Lucene (8) als Indexer und ist in C# für Mono geschrieben. Für die Kommunikation zwischen Prototyp und Beagle existieren mehrere mögliche Ansätze.

Um die Beagle-Suche zu nutzen, reicht die Nutzung des Kommandozeilentools *beagle-query* aus. Diesem kann ein Suchstring übergeben werden. Als Antwort wird eine Liste der zutreffenden Dateien generiert. Nach dem Übersetzen dieser Liste in die intern genutzten IDs kann diese direkt als Filter für die Ähnlichkeitssuche genutzt werden.

inotify Weiterhin kann der inotify Mechanismus der Beagle Engine direkt genutzt werden, um zeitnah Updates des Suchindex vornehmen zu können. Sobald eine Datei erstellt oder verändert wird, nimmt Beagle diese sofort mit in den eigenen Index auf. Um diese Veränderung auch dem Prototypen mitzuteilen, gibt es verschiedene Ansätze, die im Verlauf des Projektes noch nicht vollständig getestet wurden.

Beagle kann durch externe Filter erweitert werden. Ein Filter wird normalerweise genutzt, um Textinformationen aus dem jeweiligen Dateityp zu extrahieren und überflüssige Formatierungen oder Informationen zu ignorieren. Es ist ein spezieller Notify-Filter denkbar, der an Stelle dieser Extraktion den Dateinamen direkt an den Hintergrundprozess von Golden Retriever schickt. Dieser kann dann automatisch die neue Datei ebenfalls indexieren.

Eine Alternative wäre die Auswertung der Logfiles von Beagle. Diese Lösung hat den Vorteil, dass sie keine zusätzliche Arbeit auf der Seite von Beagle erfordert und zudem keine

eigene Suche nach neuen Dateien erfordert. Lediglich das Polling auf der Logdatei oder die Nutzung von inotify auf die Logs wären notwendig.

beaglefs Die Unterstützung einer Suche, die sich direkt auf das Dateisystem auswirkt, wurde bereits von *beaglefs* umgesetzt. Dieses Programm soll wenn möglich mit einer externen Dateiliste gefüttert werden, um die Referenzen auf die Suchergebnisse des eigenen Prototypen zu generieren. Ob dies auf einfach Weise möglich ist, konnte aus zeitlichen Gründen noch nicht näher ermittelt werden.

4.2 Probleme

Während der Arbeit am Projekt traten diverse kleine Probleme auf, die meist relativ schnell zu lösen waren. Dies betraf meist alte Unzulänglichkeiten im Programmdesign.

4.2.1 Altes Design

Ein Beispiel ist die Modellierung der GenericAccess-Klasse, die nur mit sehr wenigen Datentypen umgehen konnte und für die Speicherung der Datensätze verantwortlich ist. Die direkte Unterstützung von Integerlisten fehlte komplett, so dass eine Verarbeitung von BLOBs auf Bytebasis durchgeführt werden musste.

Für die Unterstützung eines Kategoriebaumes mussten Metatabellen mit Zusatzinformation für den Aspekt ergänzt werden, ohne das alte Konzept dabei komplett umschreiben zu müssen. Dazu wurden die Tabellen durch eine Sub-ID ergänzt, die beliebig viele Metatabellen für einen Aspekt zulässt.

Die abstrakten Klassen für die Persistenz waren alle auf den Einsatz von Datenbanken optimiert. Eine Erweiterung zum Parsen von Parameterstrings musste ergänzt werden.

Ferner musste die gesamte Verwaltung der Aspekte von variablen IDs auf Strings umgesetzt werden, da diese eindeutig sind und bei der Verarbeitung von URLs einfacher handhabbar sind.

4.2.2 Blades

Die Verwendung von Blades nahm einen nicht unerheblichen Teil der Projektzeit in Anspruch. Es traten diverse Effekte auf, die spezifisch für Serverhardware sind und bei einem einfachen Desktop System so nicht vorkommen.

Betriebssystem Die ersten Versuche mit SLES 9 waren erfolglos, da die Installation ständig mit einem unverständlich formulierten Fehler abgebrochen ist. Selbst das Aufspielen neuer Updates auf die Blade-Hardware war ergebnislos.

Die Verwendung von SLES 10.1 war schließlich erfolgreich. Vermutlich wurde dieses Problem durch eine Hardwareinkompatibilität mit den älteren Treibern hervorgerufen. M. Kabalkin hat außerdem noch einige andere Distributionen auf den Blades getestet.

4.2.3 Hardwarekonfiguration

Ein weiteres Problem war, dass die Blades nicht im Netzwerk zu finden waren. Tests haben ergeben, dass sie sich untereinander über den integrierten Switch im Bladecenter finden konnten, aber die Kommunikation nach außen nicht möglich war.

Über die Konfiguration des Managementmoduls wurde der Switch neu konfiguriert, so dass die 4 externen Ports freigeschaltet sind. Trotzdem blieben die internen Blades vom externen Netzwerk getrennt. Erst mit Hilfe eines Spezialisten konnte das Problem gelöst werden.

Die internen Ports des Switches sind standardmäßig auf das VLAN 2 gelegt und die externen Ports haben keine Zuweisung. Dadurch sind beide Netze innerhalb des Switches logisch voneinader getrennt. Für die korrekte Kommunikation muss mindestens ein externer Port ebenfalls dem VLAN 2 zugewiesen werden. Details hierzu finden sich in Anhang A.

5 Projektergebnis

5.1 Erreichte Ziele

Die Kernziele des Projektes wurden erreicht. Einige der Teilziele konnten allerdings aus zeitlichen Gründen leider nicht mehr im vollen Umfang realisiert werden.

- Die Nutzung des Dienstes ist über Servlets möglich.
- Die Software ist auf den Blades lauffähig.
- Die Unterstützung von Kategorien wurde implementiert.
- Mehrere Konzepte wurden auf ihre prinzipielle Machbarkeit überprüft.
  - Die Suche ist prinzipiell auch auf andere Daten als Bildern möglich.
  - Einbindung von Beagle als Textsuche und Notifier für Änderungen in den Daten.
  - Implementierung des *Virtual Folder* Konzepts.

5.2 Ausblick

Die zukünftige Ziele sind hauptsächlich die Fertigstellung bisher nicht oder nur zum Teil vorhandener Features.

Die Unterstützung eines Lustre Dateisystems als Basis für eine verteilte Suche wird ergänzt, sobald die Grundlagen dazu vorhanden sind. Hier wird außerdem angestrebt, neu hinzukommende Datenquellen dynamisch in die Verzeichnisstruktur einzubinden um einen einfachen Zugriff auf die Daten zu haben.

Die neu hinzugekommenen Aspektmodule bedürfen noch einer weiteren Überarbeitung. Insbesondere das Wavelet-Modul benötigt einige zusätzliche Voraussetzungen, um fehlerfrei laufen zu können.

Die bisher implementierte Servlet-Schnittstelle bedarf einer genauen Überprüfung auf Fehler oder Unstimmigkeiten. Insbesondere das Parsen eines nicht vollständigen Anfragestrings kann unter Umständen zu unerwarteten Ergebnissen führen. Eine robustere Verarbeitung ist hier notwendig.

Das Einbinden von Beagle wurde vorerst zurückgestellt, da dies die Entwicklung unter Linux zwingend erforderlich macht, aber vorerst kein geeignetes Gerät für die Entwicklung zur Verfügung stand. Selbiges gilt für die Idee der *Virtual Folder*.

Sobald die wichtigsten Funktionen vorhanden sind, wird eine gründliche Evaluierung des Systems durchgeführt, bei dem der Nutzen und die Bedienbarkeit im Vordergrund stehen.

Literatur

[1] Apache tomcat, 2007. The Apache Software Foundation. Available from: `http://tomcat.apache.org/`.

[2] Corba, 2007. Object Management Group, Inc. Available from: `http://www.omg.org/gettingstarted/corbafaq.htm`.

[3] Eclipse, 2007. The Eclipse Foundation. Available from: `http://www.eclipse.org/`.

[4] Google desktop search, 2006. Available from: `http://desktop.google.com`.

[5] Internet communications engine, 2007. ZeroC, Inc. Available from: `http://www.zeroc.com/ice.html`.

[6] itunes, 2007. Apple. Available from: `http://www.apple.com/de/itunes/`.

[7] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast multiresolution image querying. *Computer Graphics*, 29(Annual Conference Series):277–286, 1995. Available from: `citeseer.ist.psu.edu/jacobs95fast.html`.

[8] Lucene, 2007. Apache Software Foundation. Available from: lucene.apache.org.

[9] Monodevelop, 2007. Available from: http://www.monodevelop.com/Main_Page.

[10] Raoul Pascal Pein. Multi-Modal Image Retrieval - A Feasibility Study. Diplomarbeit.

[11] Java remote method invocation (java rmi), 2007. Available from: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

[12] Joe Shaw. Beagle desktop search, 2006. Available from: http://beagle-project.org/Main_Page.

[13] Novell suse linux enterprise 10 - sles 10, 2007. Available from: http://www.sles10.thecampus.de/.

[14] Xampp, 2007. Apache Friends. Available from: http://www.apachefriends.org/de/xampp.html.

A Bladecenter-Switch

Der Switch kann über den Administrationsport der Managementmoduls (192.168.70.125) konfiguriert werden. Der Zugriff ist möglich über das Webinterface des Managementmoduels oder auch direkt (192.168.70.127). Das Webinterface erlaubt grundlegende Einstellungen, sowie die Fresichaltung der externen Ports.

Die VLAN Einstellungen sind nur über eine telnet/ssh-Verbindung möglich. Der Standardaccount für die Anmeldung ist auf einem Aufkleber des Chassis zu lesen (Achtung: im Passwort ist eine Null enthalten, Kein Buchstabe O). Danach können die Ports über die folgenden Befehle umgestellt werden:

```
 switch # configure terminal

(config)# interface gigabit 0/17
(config)# switch port access vlan 2
(config)# end

(config)# write memory
```

Die externen Ports des Switches haben die Bezeichnungen 0/17 bis 0/20. Jeder Port kann einzeln auf ein VLAN eingestellt werden. Mit *write memory* werden die Änderungen dauerhaft gespeichert.