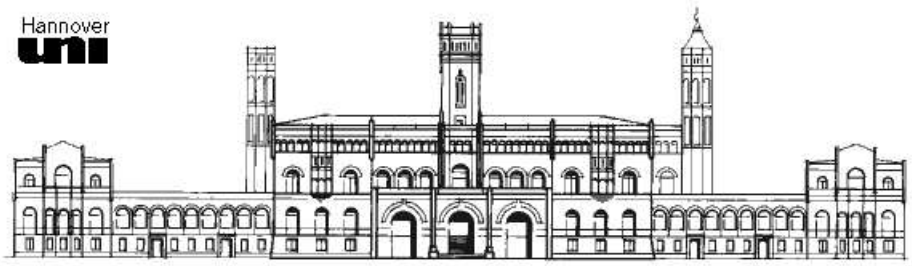

Universität Hannover

Institut für Informationssysteme
Fachgebiet Wissensbasierte Systeme

Bachelorarbeit

Erweiterung des XML Web Development Frameworks Cocoon zu einem Webservice am Beispiel der Registrierung von Primärdaten

Verfasser : Jan Hinzmann

Erstprüfer : Prof. Dr. techn. Wolfgang Nejdl
Zweitprüferin : Prof. Dr. Nicola Henze
Betreuer : Dipl.-Math. Jan Brase

Zusammenfassung

Im Rahmen des Projektes „Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten“ des Forschungszentrums L3S wurde ein Dienst zur Registrierung und Wartung dieser Daten mit Hilfe des „web development framework“s Cocoon realisiert. Ein Browserinterface , das vier Anwendungsfälle abdeckt, wurde bereits entwickelt. Zusätzlich wird eine Methode (die Registrierung von sog. DatenDOIs) über eine Web Service-Schnittstelle angeboten und somit die programmatische Registrierung ermöglicht. Auf diese Weise wurden bereits 200.000 Daten-DOIs registriert.

Der Web Service ist bisher so realisiert, dass Anfragen an das System durch Cocoon an eine integrierte Axis-Komponente weitergeleitet werden, welche dann den auszuführenden Code aufruft. Da aus der Axis-Komponente heraus nicht auf den Cocoon-internen Code, beispielsweise für Transformationen, zugegriffen werden kann, ist es momentan nur möglich die oben erwähnte Methode anzubieten. Der für die anderen Methoden existierende Code kann nicht wiederverwendet werden und es müsste für die Realisierung der anderen Methoden auf andere Technologie zurückgegriffen werden. Die daraus resultierende Verdoppelung von Code würde zu einer schlechten Wartbarkeit des Systems führen.

Ziel dieser Bachelorarbeit ist es deshalb, den Web Service durch reine Cocoon-Technologie zu verwirklichen, sodass der vorhandene Code wiederverwendet werden kann und somit die Axis-Komponente obsolet wird. Daruch soll die Wartbarkeit und Performanz signifikant erhöht werden.

Inhaltsverzeichnis

1 Einleitung **5**
- 1.1 Motivation . . . 5
- 1.2 Arbeitsfeld . . . 5
- 1.3 Überblick . . . 6

2 Projekthintergrund **7**
- 2.1 Motivation des Projektes . . . 7
- 2.2 Projektbeschreibung . . . 8
  - 2.2.1 Digital Object Identifier – DOI . . . 9
  - 2.2.2 Unified Resource Name – URN . . . 10
  - 2.2.3 Das Format für den TIB-Katalog – PICA . . . 11
- 2.3 Aufgaben des Dienstes . . . 12
  - 2.3.1 registerCitationDOI . . . 14
  - 2.3.2 registerDataDOI . . . 15
  - 2.3.3 updateURL . . . 16
  - 2.3.4 updateMetadata . . . 16
  - 2.3.5 transformData2CitationDOI . . . 17
- 2.4 Schnittstellen des Dienstes . . . 18
  - 2.4.1 Browser-Schnittstelle . . . 18
  - 2.4.2 Web Service-Schnittstelle . . . 23

3 Technologien **24**
- 3.1 Applicationserver . . . 25
  - 3.1.1 https . . . 25
  - 3.1.2 Zertifikate . . . 26
- 3.2 XML – eXtensible Markup Language . . . 27
- 3.3 XSL – Transformation von XML-Dokumenten . . . 30
  - 3.3.1 XPath . . . 31
  - 3.3.2 Einige Sprachelemente von XSL . . . 32
  - 3.3.3 XSL-Stylesheets . . . 32
  - 3.3.4 Beispiel einer (XSL)-Transformation . . . 33
- 3.4 Cocoon – Das XML Webdevelopment Framework . . . 35
  - 3.4.1 Sitemap.xmap . . . 37

3.4.2 Das Autorenbeispiel in Cocoon 40
3.4.3 Control Flow 42
3.5 Handlesystem 46
3.6 Axis – Eine SOAP-Implementation 48
3.7 Das SOAP-Protokoll 49
3.7.1 Aufbau einer SOAP-Nachricht 50
3.7.2 Möglichkeiten der Fehlerbehandlung 52
3.7.3 Die Bindung an HTTP 53

4 Die SOAP-Schnittstelle (Web Service) 55
4.1 Definition Web Service (W3C) 56
4.2 COcoonDATA – Der Web Service mit Cocoon 57
4.2.1 Allgemeines 58
4.2.2 Empfang des SOAP-Envelopes 59
4.2.3 Generierung des Methodennamens 60
4.2.4 Generierung der Parameter 62
4.2.5 Verarbeitung der Methode 64
4.2.6 Senden der Antwort 66
4.2.7 JXTemplate 68
4.3 Erweiterbarkeit 71

5 Performanztest 73
5.1 Messung 73
5.2 Resultat 73

6 Zusammenfassung 75
6.1 Bewertung und Status 75
6.2 Ausblick 76

Literaturverzeichnis 78

A Programm-CD 80

B Erklärung 81

C Danksagung 82

Kapitel 1

Einleitung

1.1 Motivation

Im Rahmen des Projektes „Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten“ wurde ein Dienst zur Registrierung und Wartung dieser mit Metadaten versehenen Datensätze realisiert. Dieser Dienst soll dem Benutzer zum einen ein Browserinterface anbieten, welches vollständig entwickelt ist und zum anderen eine Web Service-Schnittstelle für das programmatische Ansprechen des Dienstes bereitstellen. Diese SOAP-basierte Web Service-Schnittstelle vollständig auszubauen und die Wartbarkeit des Gesamtsystems signifikant zu erhöhen, ist Ziel dieser Bachelorarbeit.

Das Webinterface ist mit Hilfe des *web development framework - cocoon*[11] realisert, welches keine offensichtliche Möglichkeit bietet, ein vorhandenes Browserinterface als Web Service zu exponieren. Eine Recherche in der COCOON-eigenen Entwickler- und Benutzer-Mailingliste, der Dokumentation und dem COCOON-eigenen-Wiki führte zu keinem befriedigenden Ergebnis. Zur Zeit wird deshalb eine Standardlösung (AXIS[13]) zur Abwicklung von SOAP[14]-Verkehr benutzt, mit der es aber nicht möglich ist, den existierenden Code wiederzuverwenden. Dies hat zu einer Replikation des Codes geführt, was sich in einer schlechten Wartbarkeit und daraus resultierenden hohen Fehleranfälligkeit ausdrückt.

1.2 Arbeitsfeld

In wissenschaftlichen Arbeiten gewonnene Primärdaten können im Katalog der Technischen Informationsbibliothek registriert werden.
Dazu werden sie mit Metadaten versehen, welche alle bibliographischen Informationen (ISO 690-2) enthalten. Hierzu ist es nötig, die von den Wissenschaftlern im XML-Format gelieferten Metadaten in das PICA-

Format umzuwandeln, welches dann vom Bibliothekskatalog verstanden wird. Zur Transformation von XML-Daten bietet sich die Nutzung eines XSLT-Prozessors an, der mit Hilfe passender Stylesheets das gewünschte Ausgabeformat erzeugen kann. Ein Aspekt des Dienstes ist also die Transformation von XML-Dokumenten in andere Ausgangsformate. Weiterhin ist es nötig, generierte Ausgangsdaten auf verschiedene Arten weiterzuleiten. So akzeptiert der Gemeinsame Bibliotheksverbund [4] die gelieferten Daten im PICA-Format als FTP-Upload auf einen bestimmten FTP-Server, von wo aus die Daten in den Bibliothekskatalog eingespielt werden. Bei der Registrierung von DOIs muss hingegen eine Java^{TM}-Schnittstelle angesprochen werden, die eine durch XSL-Transformation generierte Batch-Datei benötigt. Bei der Registrierung von URNs wird eine E-Mail mit ebenfalls durch XSLT generiertem Anhang verschickt.

Die Aspekte, die bei diesem Dienst eine Rolle spielen, sind also im wesentlichen die Transformation von Eingangsdaten zu Ausgangsdaten und die anschließende Weiterverarbeitung. Diese stellt sich durch verschiedene Aufgaben dar, wie zum Beispiel einem E-Mailversand oder einem FTP-Upload.

Das Web Development Framework Cocoon [11] bietet einfache Mechanismen, eine solche Transformation zu bewerkstelligen und lässt den Entwickler durch das Konzept der Flusskontrolle Seiteneffekte nutzen; zum Beispiel zum Versenden von E-Mails.

1.3 Überblick

In den folgenden Kapiteln wird zunächst das Projekt *Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten* vorgestellt. Die Umsetzung dieses Projektes publiziert wissenschaftliche Primärdaten, in dem die Metadaten in den Bibliothekskatalog aufgenommen werden. Die Zitierfähigkeit der Datensätze wird durch den Einsatz von persistenten Identifizierern erreicht, die in den entsprechenden Systemen (DOI/URN) registriert werden. Anschließend wird auf die einzelnen Technologien eingegangen, die bei diesem Projekt zum Einsatz kommen. Hierbei wird das XML Web Development Framework Cocoon besonders im Fokus stehen. Im sich anschließenden Hauptteil der Arbeit steht die Realisierung der Web Service-Schnittstelle, bei der das SOAP[14]-Protokoll zum Einsatz kommt, im Mittelpunkt. Schließlich folgt eine Zusammenfassung und ein Ausblick auf zukünftige Arbeiten.

Kapitel 2

Projekt Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten

Das Projekt „Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten“ wird von der der TIB[2] Hannover in Zusammenarbeit mit den führenden deutschen Datenzentren aus dem Bereich der Geo-Wissenschaften durchgeführt. Es wird von der Deutschen Forschungsgemeinschaft (DFG) gefördert und ermöglicht beispielhaft am Bereich der Geo-Wissenschaften erstmalig die automatisierte Registrierung von wissenschaftlichen Primärdaten unter Verwendung von persistenten Identifizierern und Metadaten. Das Forschungszentrum L3S[1] hat die technische Betreuung des Projektes übernommen. In diesem Kapitel wird das Projekt vorgestellt.

2.1 Motivation des Projektes

In wissenschaftlichen Arbeiten werden sogenannte Primärdaten gewonnen und ausgewertet. Die ausgewerteten Rohdaten fließen anschließend in die Arbeiten der Wissenschaftler ein. Die klassische Form der Verbreitung der gesammelten wissenschaftlichen Erkenntnisse ist die Veröffentlichung in Fachzeitschriften, was in der Regel ohne die Gesamtheit der Primärdaten geschieht. Dabei wäre es für die interdisziplinäre Nutzung wünschenswert, einen permanenten Zugriff auf diese Daten zu ermöglichen. Die Bereitschaft hierzu ist prinzipiell vorhanden, allerdings wird der Mehraufwand, der für die Aufbereitung, die Kontextdokumentation und die Qualitätssicherung nötig ist, derzeit nicht anerkannt (keine Aufnahme in den sog. *”Citation Index”*).

Dies führt dazu, dass die Daten schlecht dokumentiert und nur einem kleinen Kreis von Wissenschaftlern bekannt sind. Sie sind über die einzelnen Forschungsinstitute verstreut und sind schlecht zugänglich. So bleiben große Datenbestände ungenutzt.

Auf eine Initiative des *Commitee on Data for Science and Technology* (CODATA) zur Verminderung dieser bestehenden und aufkommenden Einschränkungen in der Datenverfügbarkeit hin hat die DFG[3] das Projekt *Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten* ins Leben gerufen. Ziel ist es, die persönliche Motivation der Wissenschaftler zu erhöhen, in dem die Primärdaten eine eigene Identität erhalten und dadurch zitierfähig werden.

In Zusammenarbeit mit den Datenanbietern, dem L3S und der TIB Hannover ist deshalb ein Projekt zur Registrierung der Primärdaten entstanden, welches im Folgenden näher beschrieben wird.

2.2 Projektbeschreibung

Um den Registrierungsprozess von Primärdaten zu automatisieren, wird ein Registrierungsdienst erstellt, der die vom Wissenschaftler gelieferten Daten verarbeitet, die zur Registrierung nötigen Schritte ausführt und schließlich eine Rückmeldung über den Verlauf der Registrierung gibt. Der Wissenschaftler kann innerhalb kürzester Zeit seine Datensätze registrieren und verfügbar machen. Im wesentlichen gibt es zwei Konzepte von persistenten Identifizierern zur Registrierung der Primädaten. Dies sind die Digital Object Identifier (DOI) und die Unified Resource Names (URN). Die TIB Hannover stellt das Projekt auf hrer Homepage folgendermaßen dar:

> Ziel der exemplarischen Implementierung ist, praktische Erfahrungen bei der Einführung und im Umgang mit dem vorgeschlagenen System „Primärdaten- DOI / URN“ in der Verantwortung der Wissenschaften zu erhalten und Entscheidungsgrundlagen für eine breite Einführung im Bereich der Wissenschaft zu gewinnen. Veröffentlichungen von Primärdaten sollen als Publikation zitierbar sein, so dass der Datensatz in weiteren Verwendungen mit dem Autor zusammen genannt werden kann. Dabei werden wissenschaftliche Primärdaten nicht ausschließlich als Teil einer wissenschaftlichen Veröffentlichung begriffen, sondern sie können eine eigenständige Identität besitzen.
> Der Nachweis der Primärdaten erfolgt in TIBORDER, die Bereitstellung über die Datenzentren.

> Das Projekt wird gemeinsam vom Max-Planck-Institut für Meteorologie in Hamburg, Gruppe Modelle und Daten, dem Alfred-Wegener-Institut in Bremerhaven (Pangaea) und dem Geoforschungszentrum in Potsdam durchgeführt.

Unter dem System „Primärdaten- DOI / URN“ sind die beiden Konzepte von persistenten Identifizierern ist zu verstehen. Dies ist genauer im Abschnitt 2.3.1 auf Seite 14 zu lesen.

In einem solchen Registrierungsprozess werden die Primärdaten mit einer DOI als Indentifizierer versehen und zusammen mit einer Metadatenbeschreibung, die der ISO 690-2 zur Zitierung elektronischer Medien entspricht, zentral von der TIB verwaltet. Die eigentlichen Primärdaten als physikalische Datensätze verbleiben in den zuständigen Instituten. Diese sind auch für die Erreichbarkeit bzw. für die Aktualität der Adresse im Registrierungssystem verantwortlich. Das L3S ist für die technische Betreuung des Projektes zuständig. Die Datensätze werden im sogenannten *Handlesystem*^{TM} der *International DOI Foundation* (IDF) registriert und können dann weltweit durch sogenannte *Resolver* aufgelöst werden, wodurch sie digital zitierfähig werden.

2.2.1 Digital Object Identifier – DOI

In diesem Pilotprojekt, beispielhaft für den Bereich der Geowissenschaften, tritt die TIB Hannover seit dem ersten Juni 2005 in die Funktion einer DOI-Registrierungsagentur für Primärdaten ein. DOI-Registrierungsagenturen sind unter dem Dach der IDF organisiert und berechtigt DOIs zu vergeben und zu registrieren.

Die IDF beschreibt die DOIs wie folgt:

> The Digital Object Identifier (DOI) is a system for identifying content objects in the digital environment. DOIs are names assigned to any entity for use on digital networks. They are used to provide current information, including where they (or information about them) can be found on the Internet. Information about a digital object may change over time, including where to find it, but its DOI will not change. – (http://www.doi.org/)

Ein DOI ist also ein persistenter Identifizierer, ein Name für eine Einheit in einem digitalen Netzwerk, welcher es erlaubt, die Persistenz zu sichern. So können verschiedene Operationen, wie zum Beispiel das Auffinden eines zugehörigen Dokumentes, zu einem DOI ausgeführt werden.

Ein gültiger DOI für die Registrierung von wissenschafltichen Primärdaten könnte beispielsweise folgendermaßen aussehen:

$$10.1594/WDCC/EH4\_OPYC\_SRES\_A2$$

Für die TIB Hannover ist das Präfix 10.1594/ vorgesehen. Ihm folgt eine Bezeichnung für das entsprechende Institut, in diesem Fall für das *World Data Center for Climate* (WDCC) und abschließend ist der interne Name des Datenanbieters für die Daten zu sehen.

Ein solcher DOI kann dann weltweit durch das *Handlesystem*[7] der *Cooperation for National Research Initiatives* (CNRI)[6] mit Hilfe eines Browsers aufgelöst werden.

2.2.2 Unified Resource Name – URN

Wie die DOIs stellen auch die URNs eine Art persistente Identifizierer im Internet dar. Allerdings werden diese nicht von einer zentralen Stelle überwacht. Die Deutsche Bibliothek hat im Rahmen des Epicur-Projektes (Enhancement of Persistent Identifier Services - Comprehensive Method for unequivocal Resource Identification) eine URN-Managment-Lösung implementiert und angefangen, mit diesem System Dissertationen zu registrieren. Durch eine Kooperation der TIB und der DDB können nun auch, zunächst unter dem Namensraum (`urn:nbn:de:tib-`) der DDB, die wissenschaftlichen Primärdaten in diesem System registriert und gewartet werden. Zusätzlich hat die TIB einen eigenen Namensraum (`urn:tib:`) beantragt, welcher in Kürze genehmigt werden wird.

Neben der Registrierung über das Handlesystem wird auch jede sogenannte Zitier-DOI über das URN-System registriert. Hierbei wird der URN aus dem DOI generiert. Der DOI wird dazu mit einem bestimmten Präfix versehen und aus der so erzeugten Zeichenkette wird eine Prüfziffer errechnet. Diese wird als Suffix an die erwähnten Zeichenkette angehängt und stellt so den generierten URN dar.

Zum Beipiel ergibt sich aus dem oben erwähnten DOI mit dem Präfix `urn:nbn:de:tib-` für die Berechnung der Prüfziffer die Zeichenkette

$$urn : nbn : de : tib - 10.1594/WDCC/EH4\_OPYC\_SRES\_A2.$$

Die Prüfziffer ergibt sich in diesem Beipsiel zu 7 und somit lautet der generierte URN

$$urn : nbn : de : tib - 10.1594/WDCC/EH4\_OPYC\_SRES\_A27.$$

Die technischen Enzelheiten werden im Abschnitt 2.3.1 auf Seite 14 besprochen.

Die so registrierten URNs können momentan im Gegensatz zu den DOIs nur durch den Resolver Der Deutschen Bibliothek aufgelöst werden.

Neben der Registrierung durch die beiden persistenten Identifizierer werden die Metadaten im TIB-Katalog vorgehalten und durchsuchbar gemacht. Hierzu werden die, von den Datenanbiertern im XML-Format, gelieferten Metadaten in das PICA-Format umgewandelt.

2.2.3 Das Format für den TIB-Katalog – PICA

Die TIB setzt für ihren Bibliothekskatolg TIBORDER das Katalogschema PICA ein, welches 1967 von der Königlichen Bibliothek Holland und sechs holländischen Universitäten entwickelt wurde. PICA steht für *Project for Integrated Catalogue Automation* und basiert auf MARC II, dem klassischen System der *Library of Congress* (LoC). Die Einträge im PICA-Format bestehen aus Name-Wert-Paaren, wobei die Namen mit 4 Ziffern kodiert und in die folgenden Gruppen unterteilt sind:

0xxx *Verarbeitungsangaben* - Datum, Zeit, interne Nummer

1xxx *Kodierte Angaben* - Ercheinungsjahr, -Land, physikalische Form, etc.

2xxx *Identifikationsnummern aller Art* - ISBN, ISSN, LoC, etc.

30xx *Personennamen* - Autoren, Editoren, Widmungsempfänger, etc.

31xx *Körperschaftsnamen* - alle beteiligten Körperschaften

32xx *Sachtitel für Nebeneintragungen* - Sammlungsvermerk, Einheitssachtitel, Parallelsachtitel, etc.

4xxx *Titel Beschreibungen* - Titelinformationen inklusive Fußnoten

5xxx *Sacherschließung* - Inhaltsklassifikation, folgt verschiedenen Schemata (LoC, DDB, etc.)

6xxx *Lokaldaten* - lokale Sacherschließung

7xxx *Exemplardaten* - Datum und Selektionsschlüssel, Signatur, Magazinsignatur, etc.

8xxx *Exemplardaten* - Zugangsnummer, Verbuchungsnummer

9xxx *Inhaltliche Zusammenfassung*- Eine Übersicht über den Inhalt

Im Gegensatz zu MARCII haben die Einträge in PICA eine logische Struktur und es steht ein Bit mehr für die Kodierung zur Verfügung. Die heute etablierten 1300 verschiedenen Felder könnten in MARCII nicht mehr abgebildet werden. PICA ist außerdem stärker an den Bedürfnissen von Online-Magazinen orientiert.

In der ersten Phase des Projektes wurde bereits ein Registrierungsdienst auf Basis des *Web development frameworks* Cocoon realisiert. Das derzeitige Angebot verfügt über ein Webinterface, welches über gewöhnliche Browser zu bedienen ist und das gesamte Leistungsspektrum abdeckt. Man kann also Primärdaten registrieren und Metadaten oder die Adresse der Primärdaten ändern. Ferner wurde ein zweites Webinterface in Form eines Web Services begonnen.

Zunächst werden nun die allgemeinen Aufgaben des Registrierungsdienstes besprochen und danach wird auf die beiden Web-Schnittstellen eingegangen.

2.3 Aufgaben des Dienstes

Als Realisierung wird derzeit ein Dienst angeboten, welcher als Webinterface über einen Webbrowser oder als Web Service über einen SOAP-Clienten (zum Beispiel Axis[13]) erreichbar ist. Bei diesem Dienst gibt der Benutzer die DOI für seine Primärdaten und die URL unter der die Daten erreichbar sind ein. Außerdem lädt er die Metadaten in Form einer XML-Datei hoch und das System führt anschließend die erforderlichen Schritte durch, um dem Benutzer schließlich eine Rückmeldung über den Verlauf der Registrierung zu geben. Dieses Szenario ist in einem Anwendungsfalldiagramm dargestellt, welches die beiden Webinterfaces sowie das Cocoon -Subsystem mit den möglichen Methoden zeigt:

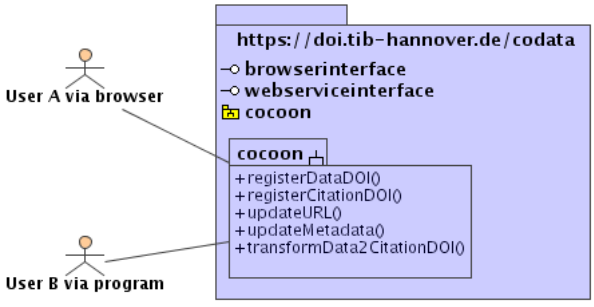

Abbildung 2.1: Anwendungsfalldiagramm (Webinterface und Web Service Klient)

Aus dem Anwendungsprofil ergeben sich die folgenden fünf Aufgaben für den Registrierungsdienst. Hierbei gibt es prinzipiell die Möglichkeit DOIs zu registrieren und registrierte DOIs zu aktualisieren. Eine Möglichkeit, Datensätze zu löschen ist nicht vorgesehen und wird in Ausnahmefällen auf Anfrage durchgeführt. Bei der Registrierung wird zwischen Zitier-DOIs und Daten-DOIs unterschieden, wobei letztere ohne Metadatenbeschreibung auskommen. Eine Wartung wird dann nötig, wenn sich zum Beispiel die Adresse des Datensatzes ändert, oder sich die Metadaten verändert haben. Schließlich können noch Daten-DOIs in Zitier-DOIs transformiert werden, für den Fall, dass sie eine Metadatenbeschreibung erhalten. Die Aufgaben des Web Services ergeben sich aus den genannten Anforderungen und es wurden folgende Methoden identifiziert:

1. registerCitationDOI – Registrierung eines Zitier-DOIs
2. registerDataDOI – Registrierung eines Daten-DOIs
3. updateURL – Erneuerung einer URL zu einer DOI
4. updateMetadata – Erneuerung der Metadaten zu einer DOI
5. transformData2CitationDOI – Transformation einer Daten-DOI in eine Zitier-DOI.

Die einzelnen Methoden sollen nun im folgenden Abschnitt beschrieben werden. Es wird absichtlich keine Methode zum Löschen eines DOIs bereitgestellt, denn ein solcher Vorgang soll nur manuell durch einen Administrator erfolgen können. Kunden können in einem solchen Fall eine E-Mail an den Betreiber schicken, welcher sich dann um eine Entfernung der entsprechenden Einträge kümmert. Dieses Prozedere verweist eindeutig dasrauf, dass ein solches Szenario eine Ausnahme darstellen sollte.

Die Anforderungen an einen Web Service zur Registrierung von Primärdaten sind vielseitig. Im wesentlichen werden als Eingabedaten DOIs, XML-Dateien (Metadaten) und URLs verarbeitet und zur Registrierung bzw. Aktualisierung herangezogen. Desweiteren ist es für die Registrierung bzw. Änderung von Einträgen nötig, die übermittelten Daten in andere Formate zu transformieren, per E-Mail oder FTP zu versenden und weitere Programme ablaufen zu lassen.

Diese Grundfunktionalitäten lassen sich am besten anhand der konkreten Methoden erläutern, welche im Folgenden vorgestellt werden.

2.3.1 registerCitationDOI

Die Methode *registerCitationDOI* stellt die umfangreichsten Anforderungen dar. Wird sie vom Benutzer gewählt, muss er eine XML-Datei mit den Metadaten zu seinen Primärdaten liefern und eine Adresse (URL) angeben, wo im Netz auf seine Daten zugegriffen werden kann. Weiter ist in den gelieferten Metadaten auch der DOI angegeben, unter dem die Primärdaten registriert werden sollen. Zuerst wird nun die Metadatendatei archiviert. Dann wird der enthaltene DOI aus den Metadaten extrahiert und zusammen mit der angegebenen URL in eine per XSL-Transformation generierte XML-Datei geschrieben.

Bei der Transformation wird aus dem, in den Metadaten enthaltenen, DOI ein URN generiert. Hierfür wird dem DOI zunächst das Präfix „urn:nbn:de:tib-“ vorangestellt und aus der so erhaltenen Zeichenkette eine Prüfziffer berechnet. Diese Prüfziffer dient als Suffix für die zuvor zusammengesetzte Zeichenkette und es ergibt sich schließlich der URN.

Die Zeichenkette, die letztendlich in dem E-Mailanhang steht, ergibt sich also beispielhaft zu: „Präfix + DOI + Suffix“.

Steht in den gelieferten Metadaten der DOI:

```
<DOI>10.1594/WDCC/GFDL_SRES_B2</DOI>
```

so erhält die Methode zur Prüfziffergenerierung diese Zeichenkette:

```
urn:nbn:de:tib-10.1594/WDCC/GFDL_SRES_B2
```

umgewandelt ergibt dies:

```
urn:nbn:de:tib-10.1594/WDCC/GFDL_SRES_B27
```

Die durch die XSL-Transformation erzeugte Datei wird anschließend als E-Mailanhang an *nbn-resolving.org* versendet, welches die Registrierung als URN darstellt.

Nun wird der DOI im Handlesystem[7] registriert. Hierzu wird eine Batch-Zeichenkette erzeugt, die einer Instanz von `Packages.net.handle.apps.batch.GenericBatch` aus dem `handle.jar` übergeben wird. Ist dies erfolgt, wird die gelieferte Metadaten-Datei mittels XSLT in das PICA-Format umgewandelt und auf einen FTP-Server des GBV [4] – Gemeinsamer Bibliotheksverbund der Länder Bremen, Hamburg, Mecklenburg-Vorpommern, Niedersachsen, Sachsen-Anhalt, Schleswig-Holstein und Thüringen – in Göttingen hochgeladen. Von dort aus wird sie weiterverarbeitet.

Der Vorgang ist noch einmal durch ein Aktivitätsdiagramm verdeutlicht (Abb. 2.2, S. 15).

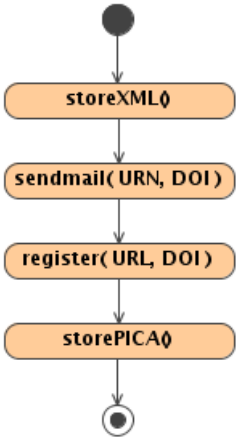

Abbildung 2.2: Aktivitätsdiagramm (Registrierung eines ZitierDOIs)

Nach Ablauf der Registrierung kann der DOI sofort durch einen sogenannten „Resolver“ aufgelöst werden. Diese Aufgabe ist vollständig automatisiert. Zusätzlich werden durch den E-Mail- und FTP-Versand der Daten noch weitere Verfügbarkeiten erreicht, was allerdings längere Zeit in Anspruch nimmt, da in diesen Fällen noch manuell oder halbautomatisch gearbeitet wird.

2.3.2 registerDataDOI

Die Methode *registerDataDOI* stellt die kleinste Methode dar. Im Gegensatz zum Zitier-DOI wird hier nur die gelieferte URL zusammen mit dem DOI im Handlesystem durch die oben beschriebene Methode registriert. Da die Methode hierfür mit einer URL und einem DOI auskommt, muss der Kunde keine XML-Datei mit Metadaten liefern. Die Angabe der beiden Parameter als Zeichenkette reicht völlig aus. Auch dieser Vorgang ist durch ein Aktivitätsdiagramm beschrieben:

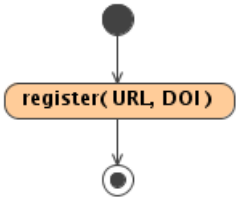

Abbildung 2.3: Aktivitätsdiagramm (Registrierung eines DatenDOIs)

Wie im obigen Fall kann ein auf diese Weise registrierter DOI sofort durch einen Resolver aufgelöst werden.

2.3.3 updateURL

Da sich Adressen zum Beispiel durch Umzug auch ändern können, ist es wichtig, eine Möglichkeit zu haben, eine URL zu aktualisieren. Eine solche Funktionalität liefert die Methode *updateURL* (Abb. 2.4, S. 16). Im Handlesystem kann dies wieder durch eine zu generierende Batchdatei mit der anschließenden Ausführung erfolgen, wonach die Aktualität wieder sofort gegeben ist. Für die Aktualisierung bei den URN-Resolvern wird ebenfalls eine E-Mail verschickt und es kommt zu höheren Latenzzeiten.

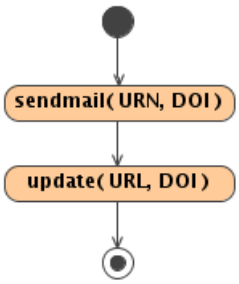

Abbildung 2.4: Aktivitätsdiagramm (Aktualisierung einer URL)

2.3.4 updateMetadata

Die Methode *updateMetadata* ist vorgesehen, wenn sich die Metadaten zu einem Primärdatensatz ändern und folglich aktualisiert werden müssen. Hierbei wird zunächst die XML-Datei lokal archiviert. Anschließend wird mittels XSL-Transformation aus der Metadatendatei eine Beschreibung im PICA-Format erstellt und diese wie bei der Registrierung von ZitierDOIs auf einen einstellbaren FTP-Server zur weiteren Bearbeitung bereitgestellt.

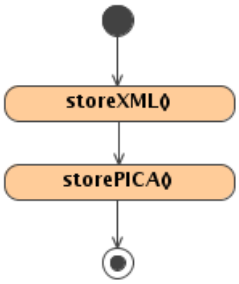

Abbildung 2.5: Aktivitätsdiagramm (Aktualisierung von Metadaten)

2.3.5 transformData2CitationDOI

Die Methode *transformData2CitationDOI* dient der Umwandlung von bereits registrierten DOI ohne Metadaten (sog. Daten-DOIs) in Zitier-DOIs, welche Metadaten haben. Hierzu wird zunächst die gelierferte Metadaten-Datei (XML-Format) archiviert. Es folgt die Transformation in das PICA-Format mit anschließendem Upload auf den FTP-Server des Gemeinsamen Bibliotheken Verbundes (GBV). Schließlich wird für die URN-Registrierung eine E-Mail mit einem Anhang verschickt, der ebenfalls durch Transformation gewonnen wird.
Prinzipiell ist das Vorgehen das Gleiche wie bei der `updateMetadata`-Methode, allerdings wird hier noch zusätzlich die URN-Registrierung vorgenommen. Eine Trennung in die beiden Methoden wurde aus Gründen der Granularität vorgenommen. Falls sich in Zukunft die Methodik ändert, bleibt so das Ausmaß der Änderungen auf den Registrierungsdienst beschränkt.

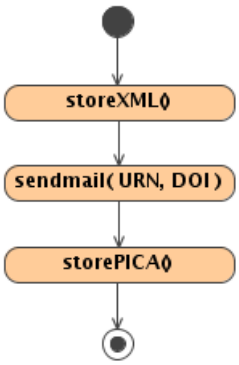

Abbildung 2.6: Aktivitätsdiagramm (Daten-DOI zu Zitier-DOI)

Nachdem nun die Aufgaben des Registrierungsdienstes dargestellt sind, sollen im Folgenden die angebotenen Schnittstellen dargestellt werden.

2.4 Schnittstellen des Dienstes

Der Registrierungsdienst für wissenschaftliche Primärdaten soll zum einen eine manuell zu bedienende Schnittstelle bereitstellen. Zum anderen soll es auch eine Programmschnittstelle geben.

Für die manuelle Bedienung bietet sich ein Browserinteface an, da die hierfür benötigte Software auf nahezu jedem Rechner vorhanden ist. Ferner gibt es freie Browserimplementierungen, die gängigen Standards folgen und es entfällt der Aufwand, eine Benutzerschnittstelle im Ganzen zu realisieren. Da diese Schnittstelle naturgemäß händisch zu bedienen ist, eigenet sie sich augenscheinlich nicht für die Registrierung großer Mengen, sondern ist als Möglichkeit gedacht, kleinere Mengen von Primärdaten manuell zu registrieren bzw. zu verwalten.

Prinzipiell kommt es zu der Situation, dass sich bei neu angegliederten Disziplinen Primärdatensätze sozusagen angestaut haben, die in einer initialen Situation in größeren Mengen (500 – 200.000) eingespielt werden sollen. Für diese Aufgabe wird eine Programmschnittstelle, welche in Kapitel 4 näher beschrieben wird, angeboten.

2.4.1 Browser-Schnittstelle

Der existierende Dienst bietet die Möglichkeit Primärdaten zu registrieren und zu warten. Dazu wird ein Webinterface angeboten, welches sich weltweit über jeden gewöhnlichen Browser bedienen lässt. Der Benutzer ruft die Seite *https://doi.tib.uni-hannover.de:8443/cocoon/codata/* auf und bekommt ein Formular (Abb. 2.7, S. 19) präsentiert, welches er ausfüllen muss und anschließend zurücksendet. Nach der Verarbeitung durch das System bekommt er eine Ergebnisseite präsentiert.

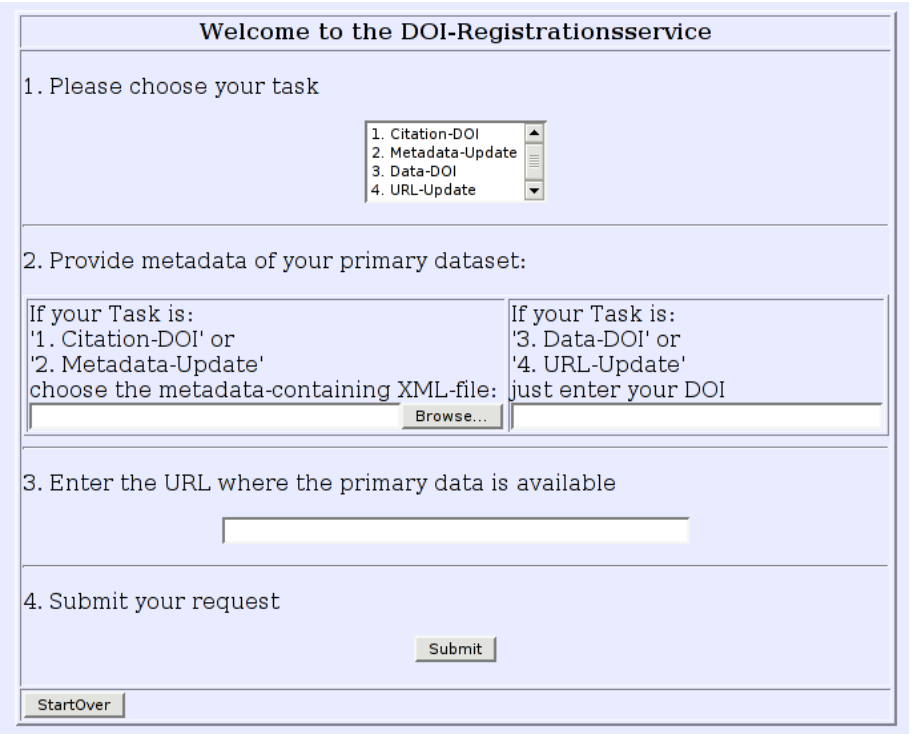

Abbildung 2.7: Das Formular des Browserinterfaces

Der Aufruf des Klienten richtet sich an eine Instanz des Sevletcontainers Tomcat, der nach einem Zertifikatsaustausch und Verschlüsselung der Verbindung, sowie der Authentifizierung mittels Benutzernamen und Passwort die Anfrage an das installierte Cocoon-Servlet weiterreicht. In der Cocoon -Instanz wird die Anfrage in einer sogenannten Pipeline verarbeitet. Dies geschiet, indem überprüft wird, ob in der Pipeline ein sogenannter *Matcher* definiert wurde, der auf die Anfrage passt. Das entsprechende Stück Code wird im folgenden Listing dargestellt:

```
<!-- == MainPipeline starts ================== -->
  <map:pipeline>
    <map:match pattern="">
      <map:redirect-to uri ="start" />
    </map:match>
<!-- Pattern to start the webinterace (flowscript) -->
    <map:match pattern="start">
      <map:call function="coDataControl" />
    </map:match>
    ...
```

In diesem Teil der Pipeline wurden zwei Matcher definiert, der erste passt auf die leere Anfrage und der zweite auf eine Anfrage nach der URL „`start`“. Die Anfrage auf einen Ordner (`.../codata/`) wird also an die Adresse `.../codata/start` weitergeleitet. Diese Weiterleitung wurde

aus Gründen der einfachen Bedienbarkeit eingeführt. An der `start`-Adresse wird die Methode *coDataControl* eines sogenanntes *flowscripts* aufgerufen. Ein Flowscirpt realisiert die Logikschicht beim MVC-Pattern (mehr dazu in 3.4.3, S. 42). Nachdem nun das Flowscript gestartet ist, werden zunächst einige Variablen initialisiert und anschließend wird mit dem Befehl „`cocoon.sendPageAndWait("requestUpload");`“ ein Formular an den Klienten zurückgesendet.

Bei diesem initialen Prozess sind gleich eine Reihe von Konzepten involviert. Zunächst ruft die Methode den Matcher mit dem Namen *requestUpload* auf. Anschließend wird das Programm angehalten und unter einem Identifizierer abgespeichert, um nach Erhalt des ausgefüllten Formulars wieder „aufzuwachen“ und an gleicher Stelle weiter zu arbeiten. Während das Programm nun sozusagen „eingefroren“ ist, wird der gerufene Matcher ausgeführt. Dies ist im folgenden Listing zu sehen:

```
1  ...
2  <!-- == Pattern to request the upload == -->
3    <map:match pattern="requestUpload">
4      <map:generate type="jx" src="xml/start.jx" />
5      <map:transform src="xsl/page2html.xsl" />
6      <map:serialize />
7    </map:match>
8  <!-- == Pattern for the continuations == -->
9    <map:match pattern="*.cont">
10     <map:call continuation="{1}"/>
11   </map:match>
12   ...
```

In Zeile 4 wird zunächst das JXTemplate *start.jx* generiert, dem die ID der Continuation mitgegeben wird. Diese wird an die Stelle der auszuführenden Aktion des Formulars gesetzt, sodass das Flowscript beim Absenden anhand des Identifizierers erkannt und deserialisiert werden kann. Anschließend wird die Ausführung an der gleichen Stelle, wie vor dem „Einfrieren“ fortgesetzt. Hierzu ist außerdem noch ein Matcher nötig, der auf alle Aufrufe passt, die mit `.cont` enden (Z.9-11 im obigen Listing).

Die so generierte XML-Datei, die das Formular (Abb. 2.7, S. 19) inklusive der Continuation-ID enthält wird nun mittels des Stylesheets *page2html.xsl* in html übersetzt und an den Klienten serialisiert.

Der Benutzer sieht dann die Tabelle mit der auszuwählenden Aufgabe, den Feldern für die URL und die DOI bzw. der Möglichkeit seine Metadatendatei auszuwählen. Er füllt die entsprechenden Felder aus und drückt schließlich auf den „Submit“ -Button. Nun wird das ausgefüllte Formular per HTTP-POST zurück an Cocoon geschickt und kommt in der Pipeline in dem oben gezeigten Matcher für die Continuations an, wo mittels `<map:call continuation="1"/>` das Flowscript wieder zum

Leben erweckt wird. Dieses generiert nun mit weiteren Aufrufen an die Sitemap zunächst die ausgewählte Aufgabe. Wenn diese bekannt ist, werden die entsprechenden Parameter aus der gesendeten Anfrage erzeugt. Sind schließlich alle notwendigen Parameter bekannt, können die enzelnen Teilaufgaben durchgeführt werden, so wie sie in Kapitel 2 beschrieben sind. Abschließend werden die von den Teilaufgaben generierten Rückgabewerte gesammelt und in einer Resümee-Seite an den Klienten gesendet.

Diese manuell zu bedienende Schnittstelle eigenet sich für die Registrierung und Wartung von wenigen Datensätzen. Wenn neue Disziplinen angeschlossen werden und ihre bestehenden Datensätze, die zum Teil in großen Mengen vorliegen (500 - 200.000 Stück), registriert werden sollen, stellt sich aber das manuell zu bedienende Browserinterface als wenig geeignet dar. Deshalb soll nun eine weitere Schnittstelle in Form eines Web Services geschaffen werden. Diese Schnittstelle wird dann programmatisch ansprechbar sein und kann ohne Probleme große Mengen von Registrierungen verarbeiten. Außerdem werden zur Realisierung bestehende Standards benutzt (XML[17], HTTP, SOAP[14], ...), sodass die Web Service-Schnittstelle, wenn sie fertig gestellt ist in die bei den Datenanbietern bestehenden Infrastrukturen eingebunden werden kann.

Das Sequenzdiagramm in Abbildung 2.8 (S. 22) fasst den gesamten Vorgang noch einmal zusammen.

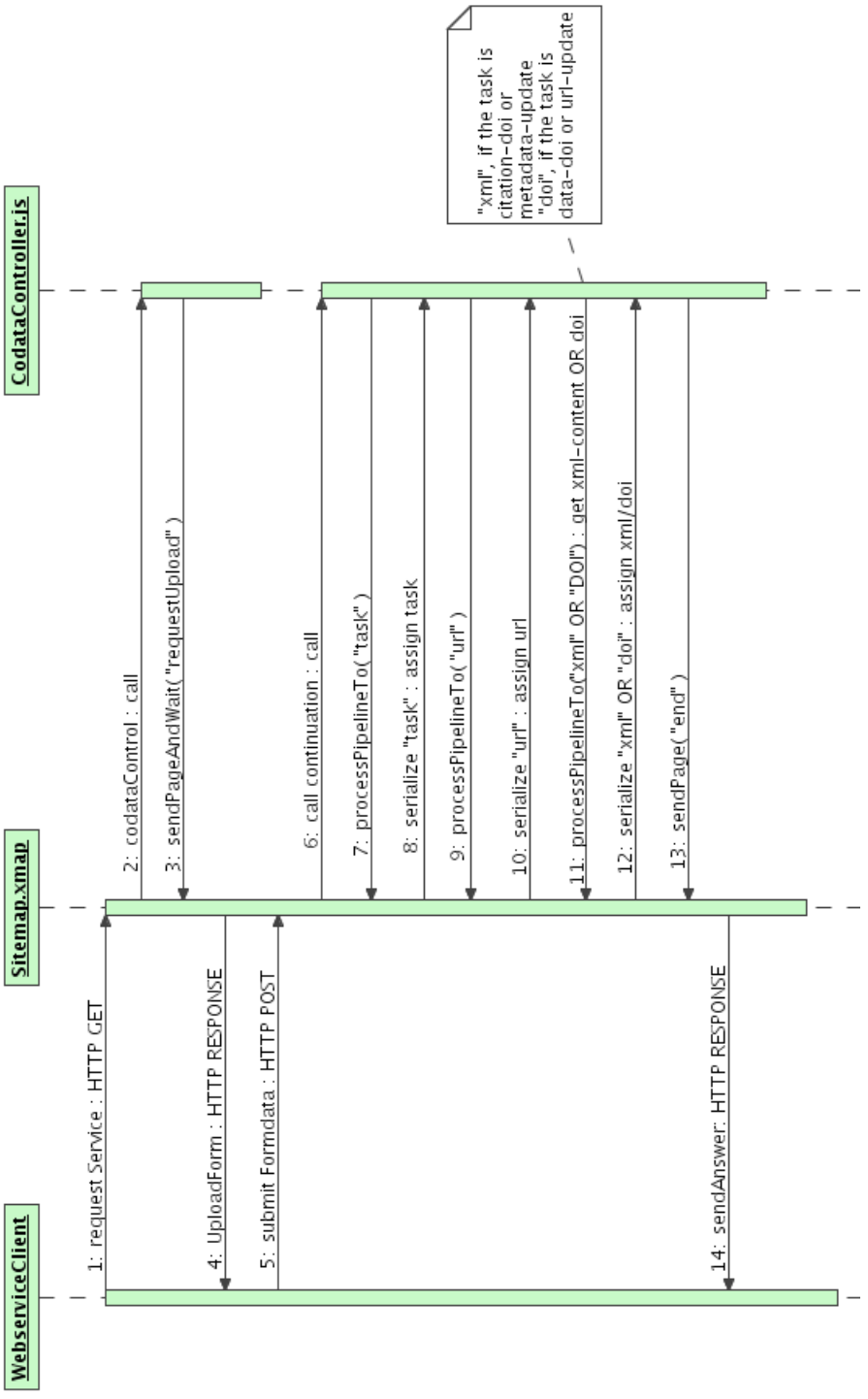

Abbildung 2.8: Sequenzdiagramm (Webinterface und Web Service Klient)

2.4.2 Web Service-Schnittstelle

Zuzüglich zu dem ersten Interface wurde ein zweites Webinterface in Form eines Web Services geschaffen, das allerdings nur eine der insgesamt fünf Methoden realisiert, die in diesem Kapitel beschrieben worden sind. Der Web Service ermöglicht es, sogenannte DatenDOIs zu registrieren, welche ohne Metadatenbeschreibung auskommen. Er versteht das SOAP-Protokoll und kann durch eine Middleware wie zum Beispiel AXIS[13] bedient werden. Mit diesem Dienst wurden bis Ende März 2005 bereits 200.000 Datensätze registriert, was die vorhandene Nachfrage verdeutlicht.
Allerdings wurde diese Funktionalität auf Kosten der Wartbarkeit erkauft, da die Web Service-Anfragen durch Cocoon an eine integrierte Axis[13]-Komponente weitergeleitet werden und es von dort aus nicht möglich ist, den vorhandenen Code wiederzuverwenden, der bereits für die Browserschnittstelle geschaffenen wurde. Für die Verarbeitung einer Anfrage musste deshalb der Code repliziert werden.
Zusätzlich ist der doppelte Code in unterschiedlichen Sprachen geschrieben (Java, Javascript), was bei Erweiterungen Übersetzungsarbeit nötg werden lässt und die Fehleranfälligkeit weiter erhöht.

Der neue Ansatz sieht nun vor, die Web Service-Schnittstelle ausschließlich mit Cooon-Technologie zu realisieren, sodass die benutzte Axis-Komponente obsolet wird. Dies erhöht auf jeden Fall die Wartbarkeit durch Vermeidung von doppeltem Code. Zusätzlich wird ein Performanzgewinn erwartet, da die Anfragen direkt verarbeitet und nicht mehr weitergereicht werden. Die Einzelheiten der Realisierung sind in Kapitel 4 beschrieben.

Im nächsten Kapitel werden nun die Technologien, die in diesem Projekt eine Rolle spielen, vorgestellt. Im Wesentlichen kommt das Cocoon-Framework in Verbindung mit dem Applicationserver Tomcat zum Einsatz. Dies stellt die vorherrschende Praxis dar. Dem Tomcat wird normalerweise aus Sicherheitsgründen noch ein Apache-Webserver vorgeschaltet, was aber erst in Zukunft realisiert werden wird.

Kapitel 3

Technologien

Dieses Kapitel widmet sich der Technologie, die bei der Realisierung zum Einsatz kommt. Im Zeitalter des aufkommenden SemanticWebs spielt die Beschreibung von Daten im XML-Format eine wesentliche Rolle und die Transformation mittels der XSL (Extensible Stylesheet Language) in beliebige andere Formate ist ein wesentlicher Bestandteil des Projektes. So liefern beispielsweise die Datenanbieter die Metadaten im XML-Format. Die TIB benötigt die Daten für ihren TIBORDER-Katalog jedoch in dem älteren PICA-Format. Dieser Konflikt lässt sich durch die XSL-Technologie auflösen. Ferner wird eine kurze Einführung in XML und XSLT gegeben, da eine grundlegende Kenntnis dieser Sprachen für das Verständnis der Webinterfaces unabdingbar ist.

Die folgende Abbildung gibt eine Übersicht über die Architektur. Wobei momentan der Anschluss an den Apache-Webserver noch nicht realisiert worden ist.

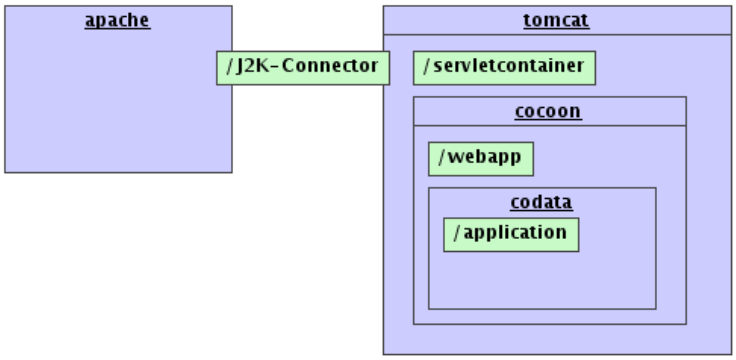

Abbildung 3.1: Architektur von CoData

Das „XML web development framework COCOON“ stellt das Herzstück der Anwendung dar. Mit seiner Pipelinestruktur und der einfach zu handhabenden XSL-Transformation ist es ideal geeignet, Daten entgegenzunehmen, sie umzuwandeln und weiterzuverarbeiten, bevor eine entsprechende Antwort an den Nutzer serialisiert werden kann.

COCOON wird in den Servlet-Kontainer *Tomcat* der Apache Software Foundation eingebettet, der Aufgaben wie die Verschlüsselung der Verbindung mittels *https* übernimmt.

Die einzelnen Technologien sollen nun im Folgenden näher beschrieben werden.

3.1 Applicationserver

Als Applicationserver kommt Tomcat zum Einsatz, welcher als *Servlet container* den HTTP-Verkehr kapselt . Tomcat ist die offizielle Referenzimplementation für die *Java*^{TM} *Servlets*- und *Java*^{TM} *Server Pages*-Technologie, die von der Firma Sun[16] entwickelt worden ist und auf deren Webseite wie folgt beschrieben wird:

> Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side–without a face. Java servlets make many Web applications possible.

Da es die Zuständigkeit des Applicationservers ist, den HTTP-Verkehr zu kapseln, fällt auch der Aspekt der Sicherheit in dieses Aufgabengebiet. Um diese zu gewährleisten, wird die Verbindung zwischen dem Klienten und dem Server über das HTTPS-Protokoll aufgebaut. So werden alle Daten verschlüsselt und ihr Austausch über die so etablierte Verbindung abhörsicher gemacht.

3.1.1 https

HTTPS steht für *hypertext transfer protocol secure* und ist ein Netzwerkprotokoll, mit dem eine gesicherte HTTP-Verbindung zwischen Rechnern möglich ist. Hierbei werden die Daten über SSL/TLS verschlüsselt und somit abhörsicher gemacht. HTTPS-Verbindungen laufen über TCP. Beim Aufbau der Verbindung findet zunächst ein Austausch geheimer Schlüssel statt, um danach in einer weiteren Stufe die jeweiligen öffentlichen Schlüssel in Form digitaler Zertifikate sicher austauschen zu können. Diese Schlüssel dienen anschließend zur Verschlüsselung der Nutzdaten, um eine abhörsichere Kommunikation gewährleisten zu können.

3.1.2 Zertifikate

Alle Anfragen werden primär an den Tomcatserver gestellt, welcher die Authentifizierung der Benutzer übernimmt. Außerdem ermöglicht der Servletcontainer eine verschlüsselte Verbindung mittels https. Zu diesem Zweck wurde mit dem Java-keytool ein Zertifikat (Abb. 3.2, S. 26) für die TIB erstellt, welches die Benutzer in ihre Liste der vertrauenswürdigen Zertifikate aufnehmen müssen.

```
Eigentumer:
    CN=TIB,
    OU=Project STD-DOI,
    O=German National Library of Science and Technology,
    L=D-30167 Hannover,
    ST=Lower Saxony, C=DE

Aussteller:
    CN=TIB,
    OU=Project STD-DOI,
    O=German National Library of Science and Technology,
    L=D-30167 Hannover,
    ST=Lower Saxony,
    C=DE

Seriennummer 42492a14

Gultig ab:
    Tue Mar 29 12:12:36 CEST 2005
          bis:
    Thu Mar 29 12:12:36 CEST 2007

Zertifikatfingerabdrucke:
MD5: 84:CF:DE:F7:33:1D:B2:C2:E2:CF:A9:EC:4D:F8:86:A6
SHA1: 9E:15:00:E9:AA:98:E0:25:8C:60:43:60:96:09:70:03:71:7D:FF:4F
```

Abbildung 3.2: Zertifikat des Dienstes zur Registrierung von Primärdaten

Nachdem die verschlüsselte Verbindung aufgebaut ist, werden noch Benutzername und Passwort verifiziert. Ab diesem Zeitpunkt werden alle Anfragen an den Tomcat direkt an das COCOON-Servlet weitergereicht und kommen dort in der sogenannten *Sitemap* an. Dies, wird im Abschnitt 3.4 näher beschrieben. Zunächst wird allerdings ein Einblick in XML und XSLT gegeben.

3.2 XML – eXtensible Markup Language

Die *eXtensible Markup Language* (XML) ist eine Teilmenge der seit 1986 standardisierten Auszeichnungssprache SGML. Ohne feste Tag-Menge erlaubt sie seinen Benutzern das Erstellen von eigenen, XML-basierten, Auszeichnungssprachen durch Benutzung von eigenen Tags und Attributen. Mit Hilfe von *Document Type Definitions* (DTD) lassen sich diese definieren.
Ein XML-Dokument besteht aus einem Prolog, einigen Elementen, die ihrerseits Attribute enthalten können und einem optionalen Epilog.

Im Prolog steht mindestens `<?xml version="1.0"?>` und optional werden hier auch Angaben über die Kodierung der Datei gemacht (z.B. `<?xml version="1.0" encoding="ISO-8859-1"?>`). Für das `encoding`-Attribut kann "UTF-8" (UTF: Unicode Transformation Format), "UTF-16" (ISO 10646 Standard), "UCS" oder "ISO-8859-1" gewählt werden. Ebenso kann mittels des `standalone`-Attributs (`<?xml ... standalone="yes/no"?>`) angegeben werden, ob das Dokument alleinstehend ist oder seine Struktur in einer externen DTD-Datei festgelegt ist. Dies wäre dann als zweites im Prolog durch beispielsweise `<!DOCTYPE book SYSTEM "book.dtd">` zu definieren. Durch `SYSTEM` wird verdeutlicht, dass es sich bei der Strukturdatei um eine lokale Datei handelt. Ist das nicht der Fall, kann eine DTD-Datei auch durch `PUBLIC` und einer URI angegeben werden.

Die XML-Elemente bestehen aus einem Anfangs- und einem End-Tag, in deren Mitte der Inhalt steht (`<foo>bar</foo>`). Die Tags können bis auf die folgenden Beschränkungen beliebig gewählt werden.

- Der erste Buchstabe muss aus {`letter|underscore|colon`} sein, also z.B. {`mytag|_mytag|:mytag`}.
- Elemente dürfen nicht mit {`xml|Xml|xMl|...`} beginnen.
- xml ist case-sensitive.

Der Inhalt eines Elements darf aus Text, weiteren Elementen oder nichts bestehen, wobei ein leeres Element, zum Beispiel `<foo></foo>` durch `<foo/>` abgekürzt werden kann.

Die XML-Attribute definieren die Eigenschaften der Elemente und sind Name-Wert-Paare innerhalb des Anfangs-Tags eines Elements, beispielsweise `<foo name="bar"/>`. Zu beachten ist hier, dass Elemente verschachtelt werden können, Attribute hingegen nicht. Zusätzlich ist die Reihenfolge der Attribute im Gegensatz zu der der Elemente beliebig.

XML-Kommentare sind von der Form

```
<!-- Hier steht ein Kommentar -->.
```

Ferner kann man noch Prozessierinformationen durch `<? ... ?>` angeben. Dies ist beispielsweise nützlich, wenn man ein Stylesheet zur Verarbeitung referenzieren will:

```
<? stylesheet type="text/css" href="./style/my.css"?>.
```

Man bezeichnet ein XML-Dokument als *wellformed*, wenn es syntaktisch korrekt geschrieben ist und als *valid*, wenn es wellformed ist und die Strukturregeln befolgt, die in einer zugehörigen Schemadatei (DTD, XSD) definiert worden sind. Die syntaktischen Regeln sind die folgenden:

- Es gibt nur ein Wurzelelement.
- Jedes Element besteht aus genau einem Anfangs- und einem End-Tag. Abkürzungen für leere Elemente sind möglich.
- Verschachtelte Tags dürfen sich nicht überlappen.
- Attribute eines Elementes müssen eindeutig sein.
- Die Elemente befolgen die oben genannten Namensrestriktionen.

Listing 3.3 zeigt eine einfache XML-Datei, die offenbar einen Studenten mit Vor- und Nachnamen ausweist, der Angewandte Informatik studiert.

```
<student>
    <name>
        <forename>Jan</forename>
        <surname>Hinzmann</surname>
    </name>
    <courseOfStudies>Informatik</courseOfStudies>
</student>
```

Abbildung 3.3: Eine simple XML-Datei

Alternativ könnte man sich den selben Sachverhalt auch durch die folgende XML-Datei vorstellen:

```
<student>
    <name first="Jan" last="Hinzmann"/>
    <studies course="Informatik"/>
</student>
```

Man sieht also, dass es verschiedene Möglichkeiten der Beschreibung gibt. Deshalb ist es wichtig, sich auf ein Format zu einigen und klare Spezifikationen zu schaffen. Dies kommt besonders bei Interoperabilität

über die Systemgrenzen hinweg zum Tragen, aber auch schon innerhalb eines einzelnen Systems mit mehreren Bearbeitern ist dies notwendig.

Für die Darstellung der XML-Dokumente ist es nötig, sie aus dem beschreibenden Format in ein Anzeigeformat (XML, (X)HTML, WML, PDF, ...) zu transformieren. Bei der Transformation kommen *Stylesheets* zum Einsatz, die in der XSL-Sprache geschrieben werden. So wird es möglich, den Inhalt vom Stil zu trennen und den gleichen Inhalt in verschiedenen Kontexten im passendem Format anzubieten. Der nächste Abschnitt wird diese Sprache näher beschrieben.

3.3 XSL – Transformation von XML-Dokumenten

Der Inhalt ist nun, mit Semantik versehen, in XML-Dokumenten abgelegt. Um diesen Inhalt anzuzeigen oder weiterzuverarbeiten, muss er in die verschiedenen Anzeigeformate (XML, (X)HMTL, PDF, WML, PICA, ...) transformiert werden. Das Werkzeug für diese Transformationen ist die *eXtensible Stylesheet Language Family* (XSL). Sie gehört zur XML-Family des W3C und setzt sich aus drei W3C-Recommendations zusammen:

XSL Transformations (XSLT) Die Sprache dient der Beschreibung, wie ein XML-Dokument in ein anderes Dokument transformiert werden soll.

XML Path Language (XPath) Diese Sprache dient der Identifikation von Elementen innerhalb eines XML-Dokumentes.

XSL Formatting Object (XSL-FO) Diese Recommendation stellt ein Vokabular für Formatierungsobjekte und Formatierungseigenschaften bereit.

Für die Transformation eines vorliegenden XML-Dokumentes in ein gewünschtes Format, sind sogenannte *Stylesheets* nötig. Diese werden zusammen mit dem Ausgangsdokumenten einem XSL-Prozessor übergeben, der den Inhalt der XML-Dokumente in verschiedenste Zieldokumente transformieren kann. Ist die Ausgabe einer solchen Transformation wieder ein XML-Dokument, so spricht man von einer *Extensible Stylesheet Language Transformation* (XSLT). Prinzipiell kann man aber verschiedenste Ausgabeformate erzeugen, wie zum Beispiel das im Abschnitt 2.2.3 von Kapitel 2 vorgestellte PICA-Format. Dieser Ablauf ist in Abbildung 3.4 dargestellt.

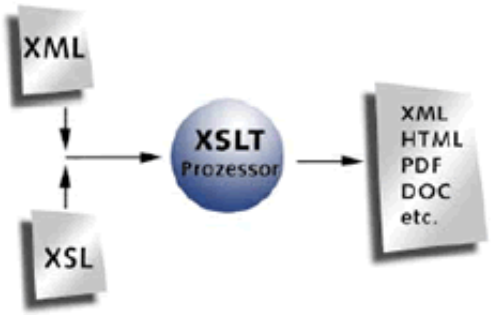

Abbildung 3.4: Prinzipieller Ablauf einer Transformation

Eine XSL-Transformation bietet prinzipiell folgende Transformationsfähigkeiten:

- Generation von konstantem Text
- Unterdrückung von Inhalten
- Verschiebung von Text
- Vervielfältigung von Text
- Sortierung
- komplexere Transformationen, die neue Informationen aus den vorhandenen errechnen/generieren

Bei der Transformation wird zunächst das XML-Dokument eingelesen und intern als Baumstruktur repräsentiert. Dieser Ausgangsbaum wird dann anhand der Regeln, welche in der ebenfalls eingelesenen XSL-Stylsheet-Datei enthalten sind, in einen Ergebnisbaum transformiert. Dieser Ergebnisbaum kann dann leicht als HTML, XML oder in anders geartetem Text serialisiert werden.

Bei der Verarbeitung von XML-Dokumenten kann im Wesentlichen zwischen zwei Konzepten unterscheiden werden. Zum einen gibt es den Ansatz des *Document Object Model* (DOM). Bei diesem Ansatz wird zunächst das gesamte XML-Dokument in den Arbeitsspeicher (RAM) geladen und dort als XML-Baum repräsentiert. Anschließend können auf dem erzeugten Baum verschiedene Transformationsoperationen durchgeführt werden und schließlich wird diese Struktur als XML-Dokument auf der Festplatte gesichert. Bei dieser Methode stellt die Größe des RAMs eine Begrenzung dar und es können nicht beliebig große XML-Dokumente verarbeitet werden.

Zum anderen steht als zweiter Ansatz die *Simple API for XML* (SAX)-Programmierschnittstelle zur Verfügung. Dieser ereignisgesteuerte Ansatz ist nicht von der Größe des zur Verfügung stehenden RAMs abhängig, da hier sequentiell auf die Elemente zugegriffen wird und immer nur der aktuelle Knoten im Speicher gehalten wird. Bei jedem eingelesenen Knoten wird ein *Event* generiert und es kann eine Verarbeitungsaktion durchgeführt werden. Anschließend wird der belegte Speicher wieder freigegeben und das nächste Element wird eingelesen.

Um in dem Ausgangsbaum navigieren, bzw. bestimmte Elemente addressieren zu können, benötigt man eine Addressierungssprache. Diese *Pfadbeschreibungssprache* ist *XPath*.

3.3.1 XPath

Mit der *XML Path Language* (XPath) lassen sich nun bestimmte Elemente aus dem Ausgangsbaum addressieren und deren Inhalt für die weitere

Verarbeitung auslesen. Die Pfadbeschreibungssprache hat eine „stringbasierte“Syntax und benutzt Pfadausdrücke, wie sie von Dateisystemen bekannt sind, für die Addressierung. Will man nun beispielsweise den Vornamen des Autors aus dem Listing 3.3 addressieren, so wäre ein gültiger Ausdruck dafür `/author/name/forename` oder auch `//forename`. Ein einfacher Slash kennzeichnet dabei absolute Pfade, wohingegen zwei Slashes relative Pfadangaben bedeuten.

3.3.2 Einige Sprachelemente von XSL

Für die Transformation stehen nun einige Sprachelemente zur Verfügung, die in einem XSL-Stylesheet Verwendung finden können. Im Folgenden findet sich ein kleiner Ausschnitt des Sprachumfangs:

`<xsl:text>` Der Inhalt dieses Tags wird in den Ergebnisbaum kopiert

`<xsl:value-of>` der Inhalt des über das Attribut `select="...">` zu spezifizierenden Knotens aus dem Eingabebaum wird in den Ergebnisbaum kopiert

`<xsl:attribute name="...">wert</xsl:attribute>` fügt dem nächsten Knoten das Attribut mit dem Namen aus `name` und Wert `wert` hinzu

`<xsl:if>` ein einfaches `if`, ohne `else` (`<xsl:if test="$condition">`)

`<xsl:choose>` realisiert ein ”switch” mit `<xsl:when test=...>` Statements und abschließendem `<xsl:otherwise>`

`<xsl:message>` dient zur Meldung von Fehlern

`<xsl:number>` realisiert Formatierungen von Überschriften, Listen, etc.

Alle Elemente (in einem Template), die nicht aus dem Namensraum `<xsl:...>` stammen, werden in den Ergebnisbaum kopiert

Eine zentrale Rolle bei der Transformation spielen die erwähnten Stylesheets, welche im folgenden Abschnitt erklärt weren.

3.3.3 XSL-Stylesheets

Im Wesentlichen besteht ein XSL-Stylesheet aus einer Reihe von *Templates*, die auf Elemente aus dem Ausgangsbaum „matchen“ und dann Anweisungen parat halten, was für diese Elemente in den Ergebnisbaum geschrieben werden soll. Für den Wurzelknoten eines solchen Stylesheets wird entweder das Tag `<xsl:stylesheet>` oder `<xsl:transform>` gebraucht. Diese beiden Tags werden synonym gebraucht und stammen aus dem gleichen Namensraum. Dieser wird üblicherweise durch

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
```

definiert. Das Konzept der Namensräume ermöglicht es Elemente, die Transformationsanweisungen darstellen von denjenigen zu unterscheiden, die eine Ausgabe bewirken sollen. Dazu werden die Transformationselemente durch das gewählte Präfix `xsl:` ausgezeichnet und alle anderen Elemente werden direkt in den Ergebnisbaum geschrieben.

Nachdem nun die wesentlichen Elemente, die bei einer Transformation beteiligt sind erklärt worden sind, werden die bisher vorgestellten Techniken an einem Beispiel verdeutlicht.

3.3.4 Beispiel einer (XSL)-Transformation

In folgendem Beispiel ist ein XML-Dokument zu sehen, welches durch ein ebenfalls angegebenes XSL-Stylesheet zu einem Output transformiert wird:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="authors.xsl"?>
<authors>
  <author>
    <name> Feuchtwanger </name>
    <book> Erfolg </book>
    <book> Die Geschwister Oppermann </book>
    <book> Exil </book>
  </author>
  <author>
    <name> Zweig </name>
    <book> Erziehung vor Verdun </book>
    <book> Der Streit um den Sergeanten Grischa </book>
  </author>
</authors>
```

Listing 3.1: Die XML-Datei

In dem XML-Dokument werden Autoren angegeben, die einen Namen haben und ein oder mehrere Bücher geschrieben haben. Die Titel dieser Bücher sind dann zusammen mit dem Namen des Autors in einem `author`-tag angegeben. Dieses XML-Dokument soll nun mit Hilfe des folgenden XSL-Stylesheets in ein HTML-Dokument transformiert werden.

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xsl:stylesheet
3   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4   version="1.0">
5  <xsl:template match="/authors">
6          <html>
7   <head><title>This is an example html</title>
```

```
8   </head>
9   <body>
10      <xsl:for-each select="author">
11    Author (<xsl:value-of select="position()"/>):
12    <ul>
13      <li><b><xsl:value-of select="name"/></b></li>
14      <ul>
15        <xsl:for-each select="book">
16     <li><xsl:value-of select="."/></li>
17        </xsl:for-each>
18      </ul>
19    </ul>
20      </xsl:for-each>
21     </body>
22       </html>
23  </xsl:template>
24 </xsl:stylesheet>
```

Listing 3.2: Das Stylesheet

Da das XSL-Stylesheet ein XML-Dokument ist, beginnt es mit der üblichen XML-Deklaration. Ihr folgt in Zeile 2 der Wurzelknoten für ein XSL-Stylesheet und als erster (und einziger) Kindknoten ist ein Template (Z. 5) zu sehen. Das Template matched mit einer absoluten Pfadangabe (XPath) auf den Knoten `authors`. In den Zeilen 6-8 wird der Kopf einer HTML-Datei deklariert, welche in Zeile 22 mit dem Endtag abgeschlossen wird. In den Zeilen 9-21 wird nun der Rumpf der HMTL-Datei definiert.
Hierzu wird zunächst das `<body>`-Element in das Ergebnisdokument geschrieben und anschließend werden in einer XSL-For-Schleife (Z. 10-20) alle Vorkommen des `<author>`-Elementes durchgegangen. Hierbei wird für jeden Knoten `'Author ('` gefolgt von der Position des aktuellen Knotens und `'):'` ausgegeben. Zusätzlich wird, noch innerhalb der For-Schleife, eine ungeordnete (HTML)-Liste erzeugt, die als Einträge dem Namen des Autors (Z. 13) und in einer weiteren ungeordneten Liste die Namen seiner Bücher (Z. 16) enthält. Dies wird mit einer weiteren For-Schleife (Z. 15) erziehlt.

```
<html>
<head>
<meta
  http-equiv="Content-Type"
  content="text/html; charset=UTF-8">

<title>This is an example html</title>
</head>
<body>
   Author (1):
   <ul>
<li><b> Feuchtwanger </b></li>
<ul>
```

```
<li> Erfolg </li>
<li> Die Geschwister Oppermann </li>
<li> Exil </li>
</ul>
</ul>
   Author (2):
   <ul>
<li><b> Zweig </b></li>
<ul>
<li> Erziehung vor Verdun </li>
<li> Der Streit um den Sergeanten Grischa </li>
</ul>
</ul>
</body>
</html>
```

Listing 3.3: Der Output

Die Ausgabe ist dann ein HTML-Dokument mit dem Inhalt des XML-Dokumentes und dem Stil des XSL-Stylesheet. In einem Browser sieht man dann die formatierte Ausgabe. Dieser generische Ansatz ermöglicht es auch, zu einem späteren Zeitpunkt den Inhalt zu verändern, also beispielsweise Autoren oder Bücher hinzuzufügen, zu ändern oder zu löschen, ohne den Stil zu verändern.

Diese Art der Generation von Inhalt ist in Cocoon mit wenigen Zeilen Code zu realisieren. Um Cocoon wird es im folgenden Abschnitt gehen.

3.4 Cocoon – Das XML Webdevelopment Framework

Das Projekt *Cocoon* ist ein "Opensource community project" der Apache Software Foundation, wurde 1999 gegründet und ist im März 2001 in der Version 2 erschienen. Während dieser Entwicklung hat es sich von einem reinen XML-Publishing-System zu einem vollständigen Web-Appikation-Framework entwickelt. Es wird von großen Unternehmen wie der Commerz Bank, Hewlett-Packard oder der NASA für Publishing-Zwecke eingesetzt.

Die Apache Software Foundation beschreibt Cocoon auf ihrer Webseite mit den Worten:

> Apache Cocoon is a web development framework built around the concepts of separation of concerns and component-based web development.
>
> Cocoon implements these concepts around the notion of 'component pipelines', each component on the pipeline specializing on a particular operation. This makes it possible to use a Lego(tm)-

> like approach in building web solutions, hooking together components into pipelines without any required programming.
>
> Cocoon is "web glue for your web application development needs". It is a glue that keeps concerns separate and allows parallel evolution of all aspects of a web application, improving development pace and reducing the chance of conflicts.

Im Wesentlichen kann man also die Funktionen von Inhalt, Stil, Logik und Management voneinander trennen (siehe Abb. 3.5). Das Konzept des sogenannten *separation of concerns* ermöglicht es verschiedenen Mitarbeitern (Grafikern, Programmierern, Redakteure, Manager) in ihrer eigenen Rolle zu arbeiten, ohne sich um die Belange der anderen Rollen zu kümmern. So arbeiten die Spezialisten in ihrem Fachgebiet und müssen sich nicht in neue bzw. fremde Domainen einarbeiten.

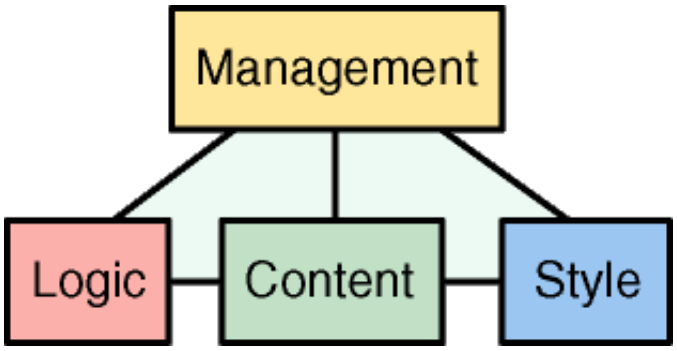

Abbildung 3.5: Cocoons Architektur des *Separation of Concerns*

Die Trennung von Inhalt und Stil wird in Cocoon durch den Einsatz von XSL-Transformation erreicht. Man legt den Inhalt in einer XML-Datei ab, die bestimmten Konventionen folgt und generiert eine Repräsentation mit einem passenden Stylesheet. Die erwähnten Konventionen können in einer DTD oder XSD-Datei festgeschrieben werden. Mit der Transformation kann man denselben Inhalt in vielen verschiedenen Arten darstellen. Am häufigsten wird dies wohl HTML oder XHTML sein, aber auch PDF oder andere Formate, zum Beispiel für mobile Geräte (Handy, PDA) stellen kein Problem dar. Prinzipiell ist jedes Format denkbar, was durch Text darstellbar ist. Ferner verfügt Cocoon über eine sogenannte *advanced flowcontrol*, wodurch es möglich wird, zusätzlich die Logik auszugliedern (siehe hierzu Abschnitt 3.4.3).

Diese Konzepte spiegeln das Model-View-Controller (MVC) - Entwurfsmuster wieder und die einzelnen Komponenten sind so stark entkoppelt, dass sie sogar in einzelne Dateien unterteilt werden können. Diese können dann von den einzelnen Rollen (Redakteur, Designer, Pro-

grammierer) bearbeitet werden. Die sich hieraus ergebenen Konsequenzen erlauben eine sehr hohe Granularität des Systems. Zum Beispiel ist es möglich, durch den Austausch der Stil-Datei das komplette Aussehen einer Anwendung zu verändern.

In Cocoon werden nun die einzelnen Informationen in der Sitemap zusammengeführt, welche das zentrale Element in einer Cocoon-Instanz darstellt. Die Sitemap wird im folgenden Abschnitt beschrieben.

3.4.1 Sitemap.xmap

Die Sitemap ist das zentrale Dokument in einer COCOON -Anwendung. Dieses XML-Dokument trägt standardgemäß den Namen `sitemap.xmap` und liegt im Wurzelverzeichnis von COCOON .

Die Sitemap wird beim erstmaligen Aufruf oder nach Änderungen in eine Java-Klasse übersetzt und steht dann innerhalb von COCOON als Objekt zur Verfügung. Entspricht das XML-Dokument nicht den Vorgaben des W3C, oder ist die Semantik fehlerhaft, so wird die Verarbeitung durch COCOON mit einer Fehlermeldung abgebrochen.

Die Struktur einer Sitemap ist prinzipiell folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
  xmlns:map="http://xml.apache.org/cocoon/sitemap/1.0">
    <map:components/>
    <map:views/>
    <map:resources/>
    <map:action-sets/>
    <map:flow/>
    <map:pipelines/>
</map:sitemap>
```

Sie stellt ein XML-Dokument dar, in dem am Anfang die Komponenten definiert und konfiguriert werden, die dann später Verwendung finden. Diese Komponenten dienen in der Regel der Erzeugung, Transformation und Serialisierung von XML-Dokumenten. Mit Hilfe der Elemente `<map:views/>`, `<map:resources/>` und `<map:action-sets/>` lassen sich definierte Komponenten gruppieren. Auf diese Gruppen kann später in den Pipelines zugegriffen werden. Das `<map:flow/>`-Element dient zur Registrierung von sogenannten *Flowscripts*, auf welche später zugegriffen werden kann. Das Konzept der Flowscripts wird im Abschnitt 3.4.3 näher beschrieben. Schließlich werden im `<map:pipelines/>`-Elemement Regeln definiert, die den Arbeitsprozess beschreiben, der bei einem Request ablaufen soll.

Die Anfragen treffen in einer Pipeline auf sogenannte *Matcher* und es werden die darin vordefinierten Anweisungen ausgeführt. Im wesentlichen ist dies immer ein dreiteiliger Prozess, bei dem zunächst ein XML-Dokument durch einen Generator als SAX-Event-Strom erzeugt wird. Dieser Strom

wird in einem zweiten Schritt an einen Transformator übergeben und geht in die XSL-Transformation ein, für die das entsprechende XSL-Stylesheet durch das `src`-Attribut angegeben ist. Das Ergebnis der Transformation wird in einem dritten Schritt an einen definierten Serialisierer übergeben, der die Antwort auf den Request erzeugt, beispielsweise eine HTML-Datei, die zum Benutzer geschickt wird und dessen Browser diese anzeigen kann. Im klassischen Fall wird auf diese Weise der Inhalt wieder mit dem Stil verknüpft und dem Benutzer als Antwort auf seine Anfrage geschickt.

Im Matcher einer Pipeline gibt es also in der Regel immer die drei Komponenten

- Generator,
- Transformator und
- Serialisierer.

Diese werden in den folgenden Abschnitten näher erklärt.

Generatoren

Ein Generator ist der Anfangspunkt einer Pipeline und generiert einen SAX-Event-Strom, den er an nachfolgende Komponenten übergibt. Jede Pipeline, die mit einem Generator beginnt, muss mit einem Serialisierer abgeschlossen werden. Es können verschiedene Generatoren im Deklarationsteil der Sitemap definiert werden, welche alle einen eindeutigen Namen erhalten müssen. Jeder Generator ist dann auf eine spezifizierte Java-Klasse gemapt und genau ein Generator wird als Standardgenerator ausgezeichnet.

Eine mögliche Konfiguration ist im folgenden Listing gezeigt, in dem ein `file`- und ein `dir`-Generator definiert werden, wobei als Standardgenerator der `file`-Generator gewählt ist.

```
...
<map:generators default="file">
  <map:generator name="file"
    src="org.apache.cocoon.generation.FileGenerator"/>
  <map:generator name="dir"
    src="MyDirGenerator"/>
  <map:generator name="serverpages"
    src="org.apache.cocoon.generation.ServerPagesGenerator">
    ...
  </map:generator>
</map:generators>
```

Cocoon bringt schon eine Reihe von Generatoren mit, die die verschiedensten Strukturen in XML darstellen und als SAX-Event-Strom an nachfolgende Komponenten weitergeben. Das Spektrum umfasst derzeit 25

Generatoren.

In der Regel schließt sich an einen Generator ein Transformator an.

Transformatoren

Transformatoren stehen in einem Matcher in der Regel zwischen einem einleitenden Generator und einem abschließenden Serialisierer. Der in den Generatoren generierte SAX-Event-Strom wird einem oder auch mehreren Transformatoren zur weiteren Verarbeitung übergeben, welche ihreseits wieder einen Strom von SAX-Events emittieren. Diese optionale Komponente taucht nach einem Generator auf und es können mehrere von ihnen hintereinandergeschaltet werden. Ebenso wie alle Sitemap-Komponenten wird auch die Transformationskomponente im Deklarationsteil der Sitemap mit einem eindeutigen Namen definiert und ihrer konkreten Java-Klasse zugeordnet. Wieder können mehrere Transformatoren definiert werden und einer von diesen muss als Standard-Komponente ausgezeichnet werden. Weitere Konfigurationsmöglichkeiten für den jeweiligen Transfomator können durch Kindelemente angegeben werden.

Für die Standardtransformation ist der XSLT-Transformator vorgesehen, der auch bei der Realisierung des Registrierungsdienstes verwendet wurde. Er liest ein im Matcher angegebenes XSL-Stylesheet ein und führt nach dessen Anleitung die Transformation durch. Dabei kann das Stylesheet auf einem lokalen Speichermedium vorgehalten werden oder über eine URL referenziert sein.

Ein XSLT-Transformer kann zum Beispiel wie folgt definiert werden:

```
...
  <map:transformers default="xslt">
    <map:transformer
      logger="sitemap.transformer.xslt"
      name="xslt"
      src="org.apache.cocoon.transformation.TraxTransformer">
    </map:transformer>
...
```

Schließlich wird durch einen Serialisierer das Ergebnis an den Klienten zurückgeschickt.

Serialisierer

Ein Serialisierer stellt den Endpunkt einer XML-Pipeline dar. Hier wird der ankommende SAX-Event-Strom in einen binär- oder char-Strom übersetzt, welcher dann zum Klienten geschickt wird. Jeder Matcher in einer Pipeli-

ne, der mit einem Generator begonnen wurde, muss mit einem Serialisierer abgeschlossen werden.

Ist ein Inhalt generiert und durch eine oder mehrere Transformationen mit seinem Stil verknüpft, wird er serialisiert und in der Regel an den Klienten zurückgesendet.

Das folgende Listing zeigt eine mögliche Konfiguration für einen XML- und einen HTML-Serialisierer, wobei der HTML-Serialisierer als Standard gewählt ist:

```
...
  <map:serializers default="html">
    <map:serializer name="xml"
      mime-type="text/xml"
      src="org.apache.cocoon.serialization.XMLSerializer"/>

    <map:serializer name="html"
       mime-type="text/html"
       src="org.apache.cocoon.serialization.HTMLSerializer"/>
...
```

Für ein Anwendungsszenario, in dem ein Internetauftritt realisiert werden soll, wäre nun wahrscheinlich bereits ein ausreichender Abstraktionsgrad erreicht. In der Sitemap einer Cocoon -Instanz wird durch die Definition der Matcher festgelegt, für welche (HTTP)-Kundenanfragen welcher Inhalt mit welchem Stil verknüpft werden und in welchem Format die Antwort serialisiert werden soll.
In den XML-Dateien steht der, zum Beispiel von Redakteuren, bereitgestellte und gewartete Inhalt ohne jede Stilinformation. Der Stil ist (physikalisch) getrennt davon in den XSL-Stylesheet-Dateien abgelegt, welche von einer anderen Gruppe von Mitarbeitern erstellt und gewartet wird. Die beschriebenen Konstrukte in der Sitemap fügen nun diese einzelnen „concerns“ wieder zusammen und der Klient erhält seine Antwort entsprechend seiner Anfrage zum Beispiel in verschiedenen Sprachen, oder passend für sein mobiles Gerät im WAP-Format.

3.4.2 Das Autorenbeispiel in Cocoon

In dem Autorenbeispiel (Abschnitt 3.3.4, S. 33) wurde gezeigt, wie mit Hilfe der XSL-Transformation ein HTML-Dokument erzeugt werden kann. Dies soll nun in eine kleinen Cocoon -Anwendung eingebunden werden.

Hierzu werden in einem Ordner `autorenbeispiel` die Dateien

- authors.xml,
- authors.xsl und

- sitemap.xmap

abgelegt. Die Dateien `authors.xml` und `authors.xsl` sind dabei die gleichen wie in dem obigen Beispiel. Der Teil der Sitemap, in dem die Pipelines definiert werden ist im folgenden Listing angegeben.

```
1  ...
2    <map:pipelines>
3      <map:pipeline>
4
5        <map:match pattern="">
6          <map:redirect-to uri="autoren" />
7        </map:match>
8
9        <map:match pattern="authors">
10         <map:generate src="authors.xml" />
11         <map:transform src="authors.xsl" />
12         <map:serialize />
13       </map:match>
14     </map:pipeline>
15
16   </map:pipelines>
17 ...
```

Weiterhin wird von einer Cocoon -Instanz ausgegangen, die auf dem lokalen Rechner installiert und unter der URL `http://localhost/cocoon` erreichbar ist.

In Zeile 2 beginnt die Definition der Pipelines und in Zeile 5 ist der erste Matcher definiert. Dieser passt auf den Aufruf `http://localhost/cocoon/autorenbeispiel/` und leitet die Anfrage an den zweiten Matcher, der auf `authors` passt weiter. Man kann also beide Anfragen stellen

- `http://localhost/cocoon/autorenbeispiel/` oder
- `http://localhost/cocoon/autorenbeispiel/authors`.

In beiden Fällen wird der zweite Matcher aufgerufen und es beginnt der eigentliche Arbeitsablauf. Zunächst wird die Datei `authors.xml` mit Hilfe des File-Generators als SAX-Event-Strom generiert (Z. 10), welcher in den sich anschließenden XSL-Transformator einfließt, der in Zeile 11 deklariert ist. Der Transformator benutzt für die Transformation das ebenfalls angegebene Stylesheet `authors.xsl`. Schließlich wird der vom XSL-Transformator emittierte SAX-Strom durch den in Zeile 12 deklarierten HTML-Serialisierer als HTML-Dokument an den Klienten serialisiert und als HTTP-Response zurückgeschickt (siehe Abb. 3.6).

Im obigen Beipiel wurde der Inhalt erfolgreich vom Stil getrennt. Für komplexere Webanwendungen und um das MVC-Pattern vollständig zu machen, ist es zusätzlich nötig, neben dem „Model“ und dem „View“ auch noch

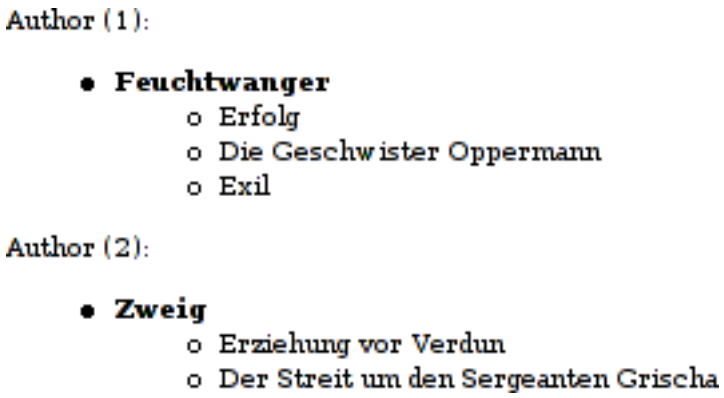

Abbildung 3.6: Die durch Cocoon produzierte Ausgabe

den „Controller“zu realisieren. Cocoon hält hier das Konzept des sogenannten „*Control Flow*“ vor, welches im nächsten Abschnitt besprochen wird.

3.4.3 Control Flow

Webanwendungen sind ereignisgesteuerte Anwendungen, die auf Benutzereingaben reagieren, in dem sie beispielsweise ihren inneren Zustand ändern und eine Antwort erzeugen. Als Konsequenz ergibt sich daraus die Notwendigkeit, die Eingaben zumindest für die Dauer der weiteren Verarbeitung persistent zu halten. In Cocoon wird dieser Aspekt durch ein sogenanntes *flowscript* realisiert. In diesem Teil der Anwendung, der in der Javascript-Sprache geschrieben ist, wird die Logik des Programms festgelegt. In einem Flowscript kann also bestimmt werden, zu welchem Zeitpunkt und in welcher Reihenfolge Seiten (Nachfrage- oder Antwortseiten) an den Klienten geschickt werden sollen.
Im Folgenden wird dies anhand eines einfachen Taschenrechnerbeipiels erläutert.

Flowscript

Das folgende Beispiel stellt einen einfachen Rechner vor, der nach Aufruf durch einen Klienten zunächst zwei Zahlen abfragt, diese daraufhin mit einem aus den vier Grundrechenarten ebenfalls auszuwählenden Operator verknüpft und das Ergebnis dieser so gewonnenen Rechnung als Antwort an den Klienten schickt. Der Aufruf des Programms erfolgt durch die Sitemap mit dem bekannten Match-Mechanismus. Hierbei muß nun zusätzlich noch das zu benutzende Flowscript deklariert werden:

```
<!-- Flow -->
  <map:flow language="javascript">
     <map:script src="calculatorController.js" />
  </map:flow>
```

In der Pipeline der Sitemap wird dann ein Matcher definiert, der auf den initialen Aufruf des Klienten passt und nichts weiter tut, als das unten angegebene Flowscript zu starten:

```
<pipelines>
  <pipeline>
    <!-- start the calculator -->
    <map:match pattern="calculator/index.html">
      <map:call function="calculator" />
    </map:flow>
```

Ruft nun ein Klient mit seinem Browser die Adresse des Rechners auf (http://.../calculator/index.html), wird die Funktion `calculator` des daklarierten Flowscripts aufgerufen. In Listing 3.7 (S. 43) ist der Programmablauf dargestellt, wie er für einen Taschenrechner gültig ist.

```
1  function calculator()
2  {
3    var a, b, operator;
4
5    cocoon.sendPageAndWait("getA.html");
6    a = cocoon.request.get("a");
7
8    cocoon.sendPageAndWait("getB.html");
9    b = cocoon.request.get("b");
10
11   cocoon.sendPageAndWait("getOperator.html");
12   operator = cocoon.request.get("op");
13
14   try {
15     if (operator == "plus")
16       cocoon.sendPage("result.html", {result: a + b});
17     else if (operator == "minus")
18       cocoon.sendPage("result.html", {result: a - b});
19     else if (operator == "multiply")
20       cocoon.sendPage("result.html", {result: a * b});
21     else if (operator == "divide")
22       cocoon.sendPage("result.html", {result: a / b});
23     else
24       cocoon.sendPage("invalidOperator.html",
25         {operator: operator});
26   }
27   catch (exception) {
28     cocoon.sendPage("error.html",
29         {message: "Operation failed: " +
30              exception.toString()});
31   }
32 }
```

Abbildung 3.7: Flowscript des Taschenrechners

In diesem Programm werden zunächst die drei Variablen *a*, *b* und *operator* deklariert (Z. 3). Anschließend werden diesen Variablen der Reihe nach durch entsprechende Interaktion mit dem Benutzer ihre Werte zugewiesen (Zeilen 6, 9 & 12). In der anschließenden `if`-Kaskade wird das Ergebnis der, mit dem ausgewählten Operator durchgeführten Rechnung, bestimmt und als Antwort an den Klienten gesendet (Z. 14-16).
Dieser Teil, bei dem mathematische Probleme (wie etwa das Teilen durch 0) auftreten können, ist zusätzlich in eine Fehlereskalation eingebettet (Z. 27-31).

Es fallen die beiden Methoden `sendPageAndWait()` und `sendPage()` auf, welche Teil des *Continuation*-Konzeptes sind. Bei der Methode `sendPageAndWait` wird eine Seite an den Nutzer geschickt und das Programm bis zur Antwort angehalten. Anschließend wird die Ausführung fortgesetzt. Die Methode `sendPage` hingegen sendet eine Seite an den Nutzer und das Programm läuft unmittelbar weiter.

In Web-basierten, also verteilten, Anwendungen sind bei der Interaktion mit dem Benutzer immer die entsprechenden Probleme, wie beispielsweise ein Verbindungsabbruch, zu beachten. Ebenso ist nicht festzustellen, wieviel Zeit der Nutzer für die Eingabe seiner Daten benötigt, in der das Programm also warten muss. Bei Browser-Schnittstellen sind ausserdem Sprünge im Programm zu beachten, wie sie etwa durch Betätigung des „Zurück„-Knopfes entstehen können.

Diesen Problemen begegnet Cocoon mit dem Konzept der sogenannten *continuations*, mit dessen Hilfe die Webanwendungen wie ein normales Programm ablaufen können. Das Konzept der Continuations wird im folgenden Abschnitt behandelt.

Continuations

Die sogenannten *continuation* stellen Objekte dar, in denen der Zustand eines Programms zu einem bestimmten Zeitpunkt festgehalten werden kann.
Durch dieses Objekt wird es möglich, ein Programm an einer beliebigen Stelle „einzufrieren“ und zu einem späteren Zeitpunkt an der selben Stelle wieder „aufzutauen“ . In Cocoon werden diese *continuation*-Objekte mit einem Identifizierer versehen und persistent gehalten.

In Listing 3.7 wurde die Methode `sendPageAndWait()` gezeigt, die Teil des *continuation*-Konzeptes ist. Das Programm hält also in Zeile 5 (Abb. 3.7, S. 43) an, nachdem es dem Klienten eine Seite mit einem Formular zur Eingabe der Zahl *a* geschickt hat. Im gesendeten HTML-Formular

steht als Aktion des *Submit*-Knopfes die URL mit dem Identifizierer des *continuation*-Objektes. So wird beim Absenden des Formulars zunächst das Programm „aufgetaut„ und in Zeile 6 kann der Variablen *a* ihr Wert aus dem nun vorhandenen Request zugewiesen werden.

Um in das gesendete Formular für die Variable *a* die ID der *continuation* einzutragen, wird ein spezieller Generator benutzt, der sogenannte *JXTemplates* generiert.

JXTemplates

Dieser Generator generiert sogenannte JXTemplates. Diese XML-Dokumente stellen generische Schablonen dar, in die Daten aus einem Flowscript eingefügt werden. Diese Daten werden als Parameter aus dem Flowscript an die Sitemap und schließlich in das Template übergeben. Er wird wie alle Sitemap-Komponenten im Deklarationsteil definiert:

```
<map:generators>
  <map:generator
    label="content,data"
    logger="sitemap.generator.jx" name="jx"
    src="org.apache.cocoon.generation.JXTemplateGenerator"/>
</map:generators>
```

In einem für die Variable passenden Matcher wird eine XML-Datei generiert, die das Formular zur Abfrage des Wertes enthält. Zusätzlich ist es in einem JXTemplate möglich, Werte dynamisch einzufügen. Hier wird nun die ID der *continuation* eingetragen.

```
<?xml version="1.0"?>
<page xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
    <title>Calculator</title>
    <content>
      <form method="post"
                  action="${cocoon.continuation.id}.cont">
        <para>
Enter value of <strong>a</strong>: <input type="text" name="a"/>
        </para>
        <input type="submit" name="submit" value="Enter"/>
      </form>
    </content>
</page>
```

Nachdem diese Datei aus dem Flowscript heraus mittels `sendPageAndWait` generiert wurde, ist der Begriff `$cocoon.continuation.id` durch die auch generierte *continuation-ID* ersetzt worden. Nun wird der SAX-Event-Strom noch mittels XSL-Stylesheet in HTML transformiert und kann an den Klienten serialisiert werden. Dieser bekommt dann Folgendes geschickt:

```
...
```

```
<h2>Calculator</h2>
<form
   xmlns:jx="http://apache.org/cocoon/templates/jx/1.0"
   method="post"
   action="26731d1988465a5d766c7f4b5235526b09160468.cont">

<p>
Enter value of <strong>a</strong>: <input type="text" name="a">
</p>
<input type="submit" name="submit" value="Enter">
</form>
</body>
</html>
```

Mit den vorgestellten Techniken lassen sich recht komplexe Webanwendungen realisieren und für weitere Informationen sei auf die „User Documentation“ (http://cocoon.apache.org/2.1/userdocs/) verwiesen.

3.5 Handlesystem

Das *Handlesystem*^{TM} ist ein, vom CNRI[6] entwickeltes, umfassendes System zur Registrierung, Wartung und Auflösung von persistenten Identifizierern. In der Einleitung auf der Homepage des Handlesystems heißt es:

> The Handle System is a comprehensive system for assigning, managing, and resolving persistent identifiers, known as "handles,"for digital objects and other resources on the Internet. Handles can be used as Uniform Resource Names (URNs).
>
> The Handle System, written in Java^{TM}and available for download at no cost for research or experimental use, includes an open set of protocols, a namespace, and an implementation of the protocols. The protocols enable a distributed computer system to store handles of digital resources and resolve those handles into the information necessary to locate and access the resources. This associated information can be changed as needed to reflect the current state of the identified resource without changing the handle, allowing the name of the item to persist over changes of location and other state information. Each handle may have it own administrator(s), and administration can be done in a distributed environment. The name-to-value bindings may also be secured, allowing handles to be used in trust management applications. (http://www.handle.net/introduction.html)

Die IDF benutzt dieses System für die Registrierung und Wartung ihrer DOIs, welche in diesem Kontext auch Handles genannt werden.

Um nun ein Handle zu registrieren bietet das Handlesystem ein Batch-orientiertes Verfahren an, bei dem eine Datei mit Befehlen eingelesen wird, um anschließend verarbeitet zu werden. Die Syntax, um eine Batch-Datei auszuführen lautet:

```
java -cp handle.jar net.handle.apps.batch.GenericBatch \
<batch> [<LogFile>]
```

In dem bestehenden Registrierungsdienst wird die Batch-Datei durch eine Transformation generiert und im involvierten Flowscript als Zeichenkette einer Instanz von `GenericBatch` übergeben. Dem zugehörigen XSL-Stylesheet (s.u.) werden als Parameter die URL und der DOI für die Registrierung übergeben

```
<?xml version="1.0" ?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" />
    <xsl:param name="url"/>
    <xsl:param name="doi"/>

    <xsl:template match="/">

AUTHENTICATE SECKEY: ***************
1594lib
CREATE <xsl:value-of select="$doi"/>
100 HS_ADMIN ***** **** ADMIN ***:***********:*.NA/10.1594
1 URL 86400 1110 UTF8 <xsl:value-of select="$url"/>

    </xsl:template>
</xsl:stylesheet>
```

Bem.: Die sensiblen Daten wurden aus datenschutzrechtlichen Gründen durch `'*'` ersetzt.

Handle Syntax

Innerhalb des Handle-Namensraumes ist jedes Handle aus einem Präfix und einem Suffix zusammengesetzt. Das Präfix wird auch als *naming authority* bezeichnet. Das Suffix besteht aus einem eindeutigen lokalen Namen. Diese beiden Teile sind durch das ASCII-Zeichen "/" getrennt. Eine (syntaktische) Definition eines Handles wäre also:
`<Handle> ::= <Handle Naming Authority> "/" <Handle Local Name>`

Das Handle „`10.1594/ba-hinzmann`“ könnte also diese Bachelorarbeit identifizieren, wobei die *Naming Authority* `10.1594` wäre und der lokale Name `ba-hinzmann`.

Handles sind *case sensitiv* es ist aber möglich, dies in dem System einzustellen, sodass ggf. darauf verzichtet werden kann.

Handle Resolution

Ist ein Handle im System registriert, kann er durch sogenannte *Resolver* aufgelöst werden. Das heißt, ein Benutzer gibt bei einem Resolver ein Handle ab und erhält daraufhin das unter diesem Eintrag abgelegte Objekt. Im Falle der DOI-Registrierung ist die DOI das Handle und die URL der Primärdaten stellt das abgelegte Objekt dar. Die Einzelheiten sind auf http://www.handle.net/overviews/overview.html nachzulesen.

3.6 Axis – Eine SOAP-Implementation

Die *Axis*-Software ist eine Implementierung des SOAP ("Simple Object Access Protocol") Protokolls. Es handelt sich um ein Framework, mit dem es möglich ist, verarbeitende Maschinen, wie Klienten, Server Gateways, etc. zu definieren, ohne dabei auf die Einzelheiten des SOAP-Verkehrs eingehen zu müssen. Die momentane Version ist in Java^{TM}implementiert und es gibt zusätzlich eine C++ Version, die die Klientenseite implementiert. Weiter bietet Axis

- einen einfachen *stand-alone server*,
- ein Servelet (zum Beipiel für Tomcat),
- gute Unterstützung für die *Web Service Description Language (WSDL)*,
- Code Generation für Java-Klassen aus WSDL,
- einige Beispielprogramme und
- ein Werkzeug, um TCP/IP-Pakete zu monitoren.

Die Axis-Software abstrahiert weitestgehend von den zugrundliegenden Techniken, sodass der Benutzer sehr einfach einen SOAP-befähigten Web Service realisieren kann, ohne sich um die Details zu kümmern.
Dazu ist es nötig, ein Java-Programm als Quelltext in einem `weapps`-Ordner abzulegen. Kommt anschließend eine Anfrage von einem Klienten, kompiliert die Axis-Technologie die Klassendefinition und ruft die entsprechende Methode auf, führt den Code aus und gibt das Ergebnis zurück. Dafür kapselt Axis den HTTP- bzw. SOAP-Verkehr.

3.7 Das SOAP-Protokoll

Das „Simple Object Access Protocol“ ist ein Kommunikationsprotokoll, das Anwendungen ermöglicht, über das Internet Nachrichtenverkehr auf eine standardisierte Weise abzuwickeln. Es basiert auf XML und ist sprach- und plattformunabhängig. Es wird als W3C-Standard (erster Working Draft erschien im Dezember 2001) entwickelt, ist einfach und erweiterbar.

Heutzutage kommen größere Anwendungen kaum noch ohne Internetanbindung aus. Also ist es wichtig, eine Kommunikation zwischen Programmen zu ermöglichen. Bei Programmen wie `COcoonDATA` bestehen wesentliche Teile der Funktionalität aus Internetanwendungen wie E-Mail- oder FTP-Versand. Viele, der derzeit verfügbaren Programme benutzen sogenannte „Remote Procedure Calls (RPC)“ zwischen Objekten wie zum Beispiel DCOM und CORBA. Das HTTP-Protokoll wurde allerdings hierfür nicht entworfen und Firewalls und Proxy-Server blockieren normalerweise diese Art von Netzwerkverkehr. SOAP ist eine neue Art von Protokoll, das über HTTP kommuniziert und von allen Internetbrowsern und -servern verstanden wird. Da SOAP auf XML basiert, bietet es eine Möglichkeit, Programme miteinander kommunizieren zu lassen, die unter verschiedenen Betriebssystemen laufen, verschiedene Technologien einsetzen, ja sogar in verschiedenen Programmiersprachen geschrieben sein können. Dies alles sind gute Gründe SOAP als Kommunikationsprotokoll zwischen Programmen einzusetzen.

Das SOAP Internet Protokoll wurde von namenhaften Firmen wie UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft, und SAP im Mai 2000 dem WWW-Consortium vorgeschlagen. Man hofft, dass es die Art der Anwendungsentwicklung durch die Verknüpfung von grafischen Desktopanwendungen mit leistungsstarken Internetservern und der Kommunikation zwischen den beiden durch SOAP bzw. die Nutzung von Standards wie XML und HTTP revolutioniert.

Wenn zwei Programme miteinander über SOAP kommunizieren wollen, verschicken sie über HTTP sogenannte SOAP-Umschläge, die wohldefinierte XML-Dokumente darstellen. Ein Programm kann also beispielsweise einem anderen Programm eine Nachricht mit der Bitte schicken, das in dem Umschlag enthaltene XML-Dokument per XSLT in ein anderes Format zu übersetzen und das Ergebnis per E-Mailversand an eine bestimmte, ebenfalls in der Nachricht angegebene Adresse zu senden. Das andere Programm empfängt diese Nachricht, verarbeitet sie und schickt schließlich eine Antwortnachricht an das erste Programm inklusive einer möglichen Fehlermeldung zurück.

Im Folgenden sollen diese Nachrichten, also die Umschläge und deren Inhalt genauer beschrieben werden.

3.7.1 Aufbau einer SOAP-Nachricht

Da eine SOAP-Nachricht ein gewöhnliches XML-Dokument darstellt, handelt es sich hier um Textnachrichten, geschrieben in einer Auszeichnungssprache, deren Tags dem Nachrichtentext seine Bedeutung verleihen. Als Wurzelknoten einer SOAP-Nachricht muss ein *Envelope*-Element vorkommen, welches das XML-Dokument zu einem SOAP-Umschlag macht. Es kann ein optionales *Header*-Element folgen, während ein *Body*-Element erforderlich ist. In diesem Bodyelement sind später Aufruf- und Antwortinformationen enthalten. Außerdem kann noch ein optionales *Fault*-Element folgen, welches Fehlermeldungen enthalten kann, die beim Verarbeiten der Nachricht aufgetreten sind. All diese Elemente werden im Standardnamensraum für SOAP-Umschläge `http://www.w3.org/2001/12/soap-envelope` deklariert. Der Standardnamensraum für Datentypen und Encoding ist `http://www.w3.org/2001/12/soap-encoding`. Als weitere syntaktische Regeln gelten, dass eine SOAP-Nachricht

- in XML geschrieben sein muss,
- den SOAP-Envelope-Namensraum benutzen muss,
- den SOAP-Encoding-Namensraum benutzen muss,
- keine DTD-Referenz enthalten darf und
- keine XML-Prozessierinstruktionen enthalten darf.

Die Abbildung 3.8 (S. 51) zeigt beispielhaft das Grundgerüst einer SOAP-Nachricht.

Das SOAP-Envelope-Element als Wurzelelement des XML-Dokumentes muss aus dem oben angegebenen Namensraum stammen, ansonsten ist es die Aufgabe der verarbeitenden Anwendung, mit einem Fehler abzubrechen und die Nachricht nicht weiter zu verarbeiten. Das Attribut `encodingStyle` kann in jedem SOAP-Element enthalten sein und wird benutzt, um die Datentypen zu definieren, die in der Nachricht vorkommen. Es wird auf den Kontextknoten (aktueller Knoten) und auf alle Kindknoten angewandt (Es gibt keine Standardeinstellung für das Encoding).

Das SOAP-Header-Element, welches optional als erstes Kindelement des Wurzelknotens auftreten kann, enthält anwendungsspezifische Informationen, wie zum Beispiel Daten über die Authentifizierung. Alle Kindknoten des Header-Elementes müssen in einem Namensraum definiert werden. SOAP kennt drei Attribute aus dem Namensraum

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  ...
  ...
</soap:Header>

<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Abbildung 3.8: Eine skeletthafte SOAP-Nachricht

`http://www.w3.org/2001/12/soap-envelope`, die im Header benutzt werden können, um dem Empfänger mitzuteilen, wie er die Nachrict verarbeiten muss. Dies sind im Einzelnen das `actor`-, das `mustUnderstand`- und das oben erwähnte `encodingStyle`-Attribut.

Das `actor`-Attribut dient dazu, Elemente im Header einer oder mehreren Stationen auf dem Weg der Nachricht zum Empfänger zuzuweisen. Eine Nachricht passiert auf ihrem Weg vom Sender zum Empfänger möglicherweise mehrere Stationen und nicht alle Informationen sind unter Umständen für den endgültigen Empfänger bestimmt.

Das `mustUnderstand`-Attribut dient der Anzeige, ob ein Eintrag im Header optional oder obligatorisch für die Verarbeitung beim Empfänger ist. Wenn man also in einem Kindelement des Headers ein `mustUnderstand=''1''` Attribut plaziert, muss der Empfänger dieses Element verstehen, ansonsten muss er die Verarbeitung mit einem Fehler abbrechen.

Das obligatorische SOAP-Body-Element enthält die eigentliche Nachricht, die für den Empfänger bestimmt ist. Direkte Kindelemente können durch einen Namensraum qualifizert werden und SOAP definiert nur ein direktes Kindelement im Standardnamensraum innerhalb des Bodys, das optionale `Fault`-Element.

Eine minimale Nachricht, in der ein Sender einen Empfänger nach dem Preis für iPods fragt, ist im folgenden Listing gezeigt:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
   <preis:getPrice
      xmlns:preis="http://jan.geeksonly.de/preise">
      <preis:Item>iPod</preis:Item>
   </preis:getPrice>
</soap:Body>

</soap:Envelope>
```

Dabei sind die Elemente innerhalb von `soap:Body` anwendungspezifisch und in einem eigenen Namensraum außerhalb des SOAP-Standards definiert.

Ein Server kann nun diese Anfrage verarbeiten und das Ergebnis im folgenden Antwort-Envelope an die anfragende Anwendung senden:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
   <preis:getPriceResponse
      xmlns:preis="http://jan.geeksonly.de/preise">
      <preis:Price>50</preis:Price>
   </preis:getPrice>
</soap:Body>

</soap:Envelope>
```

3.7.2 Möglichkeiten der Fehlerbehandlung

Das bereits erwähnte `Fault`-Element dient dazu, auch durch entfernte Aufrufe oder verteiltes Rechnen eine sinnvolle Fehlereskalation betreiben zu können. Es kommt einmalig als Kindelement des `Body`-Elementes vor und beinhaltet eine Fehlermeldung. Ein solches `Fault`-Element enthält die folgenden Subelemente

- `faultcode` – mit einem Code, zur Identifikation des Fehlers
- `faultstring` – mit einer menschenlesbaren Beschreibung des Fehlers
- `faultactor` – mit Informationen, wer den Fehler verursacht hat

- `detail` – mit anwendungsspezifischen Fehlerinformationen bezogen auf das `Body`-Element.

Für das Element `faultcode` stehen vier mögliche Fehlerarten bereit, die als Wert innerhalb des Knotens auftreten dürfen. Zum einen gibt es einen `VersionMismatch`-Fehlercode, der besagt, das ein ungültiger Namensraum für das SOAP-Envelope-Element benutzt wurde. Der `MustUnderstand`-Fehlercode kommt zum Einsatz, wenn ein direkter Nachfolger des Header-Elementes, mit dem `mustUnderstand`-Attributwert eins (also wahr) nicht verstanden wurde. Der `Client`-Fehlercode wird benutzt, wenn die Nachricht syntaktisch nicht korrekt formuliert ist oder inkorrekte Informationen enthält. Schließlich gibt es noch den `Server`-Fehlercode, der besagt, dass es ein Problem mit dem Server gab und die Nachricht nicht verarbeitet werden konnte. Eine Fehlermeldung könnte also beispielsweise folgerndermaßen aussehen:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <soap:Fault>
    <soap:faultcode>Server</soap:faultcode>
    <soap:faultstring>
      Dienst wegen Reinigung nicht erreichbar
    </soap:faultstring>
    <soap:faultactor>
      http://jan.geeksonly.de/dienst
    </soap:faultactor>
    <soap:detail>
      The service you have called,
      is temporarily not available
    </soap:detail>
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

3.7.3 Die Bindung an HTTP

Das Simple Object Access Protocol ermöglicht es also Anwendungen Nachrichten über HTTP und somit über das Internt auszutauschen. HTTP an sich kommuniziert über TCP/IP und ein Klient verbindet sich darüber mit einem Server. Ist eine Verbindung aufgebaut, kann der Klient HTTP-Anfragen (Requests) an den Server schicken. So könnte ein Kleint beispielsweise folgende Anfrage senden:

```
POST /item HTTP/1.1
Host: 130.75.152.142
```

```
Content-Type: text/plain
Content-Length: 200
```

Wenn der Server (130.75.152.142 in diesem Beispiel) diese Anfrage korrekt verarbeitet hat, schickt er eine Antwortnachricht mit einem Statuscode für die Anfrage an den Klienten zurück. Die sieht im Normalfall so aus:

```
200 OK
Content-Type: text/plain
Content-Length: 200
```

Kann der Server die Nachricht aus irgendeinem Grund nicht verstehen, kann er Folgendes antworten

```
400 Bad Request
Content-Length: 0
```

Eine SOAP-Methode ist eine Nachricht die, aus HTTP-Anfragen/-Antworten und einem SOAP-Envelope zusammengesetzt, als HTTP-POST oder HTTP-GET an einen Server verschickt, von diesem verarbeitet und als HTTP-RESPONSE inklusive dem Antwortumschlag an den Klienten zurückgeschickt wird. Es gilt also sozusagen HTTP + XML = SOAP.
Nachdem nun die benutzte Technologie beschrieben ist, wird im folgenden Kapitel das Interface des Web Services und seine Implementierung vorgestellt.

Kapitel 4

Die SOAP-Schnittstelle (Web Service)

Dieses Kapitel stellt die Methoden und Mechanismen vor, mit denen schließlich der Web Service realisiert worden ist.
Für das Projekt „Publikation und Zitierfähigkeit von wissenschaftlichen Primärdaten“ wurde unter Verwendung von Cocoon ein Dienst in der TIB etabliert, der die vorgestellten Anforderungen (Kap. 2) umsetzt. Dieser Dienst stellt, wie es für die meisten Cocoon -Anwendungen üblich ist, ein Browserinterface zur Verfügung.

Dieses Browserinterface ist naturgemäß manuell zu bedienen, allerdings stellte sich heraus, dass es wünschenswert ist, über eine Möglichkeit zu verfügen, Datensätze auch programmatisch zu registrieren. Vor dem Hintergrund der großen Menge an Datensätzen ($\geq$ 200.000), die mit dem Dienst registriert werden sollen, erscheint eine Programmschnittstelle unabdingbar. Um nun den Registrierungsdienst als Bindeglied zwischen der bestehenden Infrastruktur der Datenanbieter auf der einen Seite und dem Handlesystem, dem URN-Registrierungssystem und dem Katalog der TIB auf der anderen Seite zu integrieren, soll ein zweites Webinterface in Form eines Web Services geschaffen werden.

Idealerweise könnten dabei die Methoden wiederverwendet werden, die bereits bei der Erstellung des Browserinterfaces geschaffen worden sind. In Abbildung 4.1 (S. 56) ist der Anwendungsfall mit den beiden Webinterfaces noch einmal verdeutlicht.

Zunächst soll nun die Frage geklärt werden, was ein sogenannter Webservice ist und im Anschluss wird die geschaffene Lösung im Einzelnen besprochen.

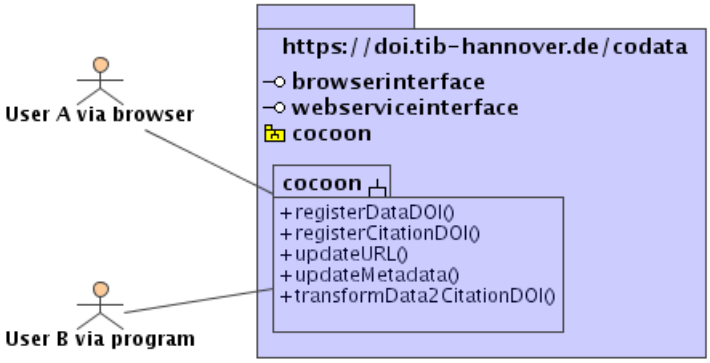

Abbildung 4.1: Anwendungsfalldiagramm (Webinterface und Web Service Client)

4.1 Definition Web Service (W3C)

Das World Wide Web wird immer mehr für die Kommunikation von Anwendungen untereinander benutzt. Dabei werden die Schnittstellen, die ein Softwaresystem im Netz anderen Programmen zur Verfügung stellt, oft als *Web Service* bezeichnet.

Prinzipiell ruft dabei ein Klient-Programm bei einem Service-Programm einen Dienst (bzw. konkret eine Methode) über definierte Schnittstellen und Protokolle auf. Das Service-Programm verarbeitet diese Anfrage, in dem es beispielsweise selbst einen anderen Dienst zu Rate zieht oder Berechnungen durchführt, und liefert ein Ergebnis als Antwort. Die Grundlage für die Web Services bilden die vier XML-basierten Standards

SOAP oder XML-RPC für die Kommunikation bzw. den Datenaustausch,

WSDL für die Definition der Schnittstelle und

UDDI als Verzeichnisdienst zur Registrierung von Web Services.

Als Übertragungsprotokoll fungiert HTTP, wodurch es selten zu Problemen mit Firewalls kommt, welche bei vergleichbaren Technologien wie CORBA, DCOM oder auch Java RMI auftreten. Die Verwendung von bekannten und weit verbreiteten Standards hat zum Vorteil, dass so offene und flexible Systeme entstehen können, die plattformunabhängig sind. Auch die Programmiersprache bleibt substituierbar, so ist es beispielsweise kein Problem, den Registrierungsdienst der TIB über ein Ruby-Skript anzusprechen. Als nachteilig erweist sich die Tatsache, dass grundsätzlich mehr Wissen für die Realisierung erforderlich ist, etwa für die Verwendung der SOAP-Technologie.

Im Bereich der verteilten Systeme stellen Web Services eine Technologie dar, mit der es ohne Probleme möglich ist, komplexe Business-to-Business (B2B) Prozesse über die Systemgrenzen hinweg zu implementieren.

Das W3C definiert den Term *„Web Service“* wie folgt:

> a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifact. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocol.” [W3C]

Im Fall des Registrierungsdienstes für wissenschaftliche Primärdaten sind die einzelnen Forderungen durch folgende Techniken realisiert:

URI Der Web Service ist unter einer URL zu erreichen. Die Verbindung ist durch HTTPS gesichert und es ist notwendig, sich mit einem Benutzernamen und einem Passwort bei dem Dienst anzumelden.

defined, described, discovered Die Web Service-Schnittstelle ist in einer sogenannten WSDL-Datei, einem XML-Dokument, beschrieben. Diese wird bei Anbindung von neuen Fachbereichen bekannt gemacht.

direct interaction via XML-based messages Die Anfragen und Antworten werden über das SOAP-Protokoll abgewickelt, einem XML-basierten Protokoll für Nachrichtenverkehr zwischen Maschinen. Dieses Protokoll wurde in Kapitel 3 im Abschnitt 3.7 erklärt.

exchanged via Internet-based protocol Der Nachrichtenverkehr wird über das internetbasierte HTTP-Protokoll abgewickelt.

Wie bereits oben erwähnt, kommt es bei der Angliederung von neuen Disziplinen zu der intitialen Situation, dass sehr viele Datensätze registriert werden müssen. Hier ist es sinnvoll, eine API anzubieten, mit der man automatisiert große Mengen verarbeiten kann. Für diese Schnittstelle wird eine Realisierung angeboten, die das SOAP-Protokoll versteht. Die Einzelheiten sollen jetzt beschrieben werden.

4.2 COcoonDATA – Der Web Service mit Cocoon

Prinzipiell ist in Cocoon kein Weg vorgezeichnet, wie man einen Web Service realisieren kann. Auch eine gründliche Recherche in der Userdocumentation, dem Cocoon -Wiki und der Mailingliste brachte keine Klärung, wie mit Cocoon ein Web Service zu realisieren ist. Es musste also ein eigenes Konzept entwickelt werden.

Wie der Web Service schließlich realisiert werden konnte, wird nun im Einzelnen beschrieben. Hierzu werden zunächst allgemeine Aspekte verdeutlicht. Im Anschluss wird der Empfang des SOAP-Envelopes beschrieben, gefolgt von der Verarbeitung bis hin zum Senden der SOAP-Antwort. Wobei für die Verarbeitung zunächst der Methodenname aus der SOAP-Anfrage generiert werden muss. Ist dieser bekannt, können die zugehörigen Parameter aus der Anfrage extrahiert und entsprechend der zugehörigen Methode verarbeitet werden.

4.2.1 Allgemeines

Stellt man sich die Frage, was ein SOAP-basierter Web Service leisten muss, so kommt man im Wesentlichen auf einen dreiteiligen Ablauf, in dem

1. ein Klient eine Anfrage (SOAP-Request) schickt,
2. das System die Anfrage verarbeitet und alle nötigen Schritte durchführt und
3. der Klient eine Antwort über den Verlauf des Auftrages (SOAP-Response) erhält.

Hierbei sind Anfrage und Antwort XML-Dokumente, die sich an das SOAP-Protokoll halten. Eine SOAP-Antwort aus Cocoon heraus zu versenden, kann leicht durch eine parametrisierte Transformationen realisiert werden.

Sieht man den gesamten Prozess als Transformation einer Anfrage in eine Antwort mit Nebeneffekten an, so würde ein eingehende SOAP-Nachricht als SAX-Event-Stream generiert werden müssen. Anschließend müsste mit einer oder mehreren Transformationen die aufgerufene Methode, sowie die eventuell übergebenen Parameter aus der SOAP-Anfrage ermittelt werden. Anschließend würde das System über alle nötigen Informationen verfügen und die erforderlichen Prozesse zur Verarbeitung der Anfragen durchführen. Schließlich würde dann das Ergebnis oder eventuelle Fehlermeldungen in einer SOAP-Antwort generiert und an den Klienten geschickt werden.

Der gesamt Prozess kann also als Transformation einer Anfrage in eine Antwort gesehen werden, bei der als Seiteneffekte bestimmte Operationen ausgeführt werden, die für die korrekte Verarbeitung der Anfrage nötig sind. Hierin spiegelt sich genau das dreiteilige Prinzip in den Pipelines der Cocoon -Sitemap wieder (s. Kap. 3.4.1).

Aus einer Cocoon -Pipeline heraus ein XML-Dokument an den Klienten zu serialisieren stellt kein Problem dar. Die Transformation ist ebenfalls einfach zu bewerkstelligen. Das Gesamtproblem kann also auf die Generierung des eingehenden SOAP-Umschlags aus dem Request

des Klienten reduziert werden. Genauer gesagt, gilt es den Namen der aufgerufenen Methode, sowie die übergebenen Parameter zu generieren.

Um dem prinzipiellen Ablauf zu verstehen, wird im Folgenden der Ablauf eines Aufrufes der Methode `registerCitationDOI` verdeutlicht.

4.2.2 Empfang des SOAP-Envelopes

Ähnlich wie beim Browserinterface tritt ein Klient, in diesem Fall allerdings ein Programm, an den Registrierungsdienst heran. Es schickt nach erfolgreichem Verbindungsaufbau mittels eines HTTP POST-Befehls einen SOAP-Envelope an den Endpunkt *https://doi.tib.uni-hannover.de:8443/cocoon/webservice*. Hinter der obigen URL verbirgt sich die gleiche Cocoon -Instanz wie sie für das Browserinterface installiert worden ist. Der gesendete SOAP-Envelope beginnt folgendermaßen:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <ns1:registerCitationDOI
          soapenv:encodingStyle=
              "http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:ns1="CodataWS">
        <ns1:arg0 xsi:type="xsd:string">
        ...
```

Auf den klientseitigen Teil der Anwendung wird in diesem Beipiel nicht näher eingegangen, da der Fokus auf der Serverseite liegen soll. Für die Seite des Klienten kann beispielsweise die Axis-Technologie[13]eingesetzt werden.

Es handelt sich um eine typische Axis-Installation, mit einem kleinen Web Service-Klienten, der die Methode `registerCitationDOI` aufruft. Diese Methode erwartet zwei Parameter, eine als `String` übergebene XML-Datei und einen zweiten Parameter, der ebenfalls vom Typ `String` ist und die zum DOI gehörende URL darstellt.

Der Klient ruft also den Matcher für `'webservice'` in der Sitemap der Cocoon -Instanz auf:

```
<!-- == Web Service by Cocoon ======== -->
  <!-- calling flow-->
  <map:pipeline>
    <map:match pattern="webservice">
      <map:call function="coDataWS" />
    </map:match>
```

Hier wird nun die Methode `coDataWS` des Flowscripts aufgerufen, welche als erstes den gesendeten SOAP-Envelope mit Hilfe der Funktion `processPipelineTo` generiert und in der Variablen `soapData` für die spätere Wiederverwendung abspeichert.

```
//getting the envelope out of the request
//(can be done only once)
var soapData = new java.io.ByteArrayOutputStream();
cocoon.processPipelineTo("soapData", null, soapData);
```

Die `processPipelineTo`-Methode der COCOON -Instanz steht innerhalb eines Flowscript zur Verfügung und hat folgende Signatur:

```
void processPipelineTo(
    [String] uri,
    [Object] bizData,
    [java.io.OutputStream] stream
    )
```

Hierbei ist `uri` der Name des aufzurufenden Matchers in der Pipeline, mit dem Objekt `bizData` können ein oder mehrere Parameter übergeben werden und in die Variable `stream` wird schließlich das Ergebnis der Pipeline geschrieben.

Der aufgerufene Matcher generiert den SOAP-Envelope aus dem HTTP-Request als Stream und serialisiert ihn gleich darauf, ohne den XML-Prolog (`xmlnope`):

```
<!-- getting the soap-envelope -->
<map:match pattern="soapData">
  <map:generate type="stream"/>
  <map:serialize type="xmlnope"/>
</map:match>
```

Ab diesem Zeitpunkt ist die erhaltene Anfrage gespeichert und es muss nun ermittelt werden, welche Methode aufgerufen wurde. Ist diese bekannt, ist auch klar, welche Parameter übergeben worden sind, wenn der Klient sich an die Spezifikationen aus dem WSDL-Dokument gehalten hat.

4.2.3 Generierung des Methodennamens

Zunächst muss nun der Name der aufgerufenen Methode aus dem empfangenen SOAP-Envelope extrahiert werden. Nach den Konventionen des SOAP-Protokolls ist der Methodenname der Name des ersten Knotens innerhalb des `<Body>`-Tags. Die Position des Namens innerhalb der SOAP-Nachricht, welche ein XML-Dokument darstellt, ist also bekannt und kann mit Hilfe eines geeigneten XSL-Stylesheets extrapoliert werden. Hierzu dient wieder die Methode `processPipelineTo(...)`, welche nun den zuvor generierten SOAP-Envelope als Parameter enthält:

```
//getting the method out of the soap-content
var soapMethod   = new java.io.ByteArrayOutputStream();
cocoon.processPipelineTo(
    "soapAsText",
    {"soapData":soapData,
      "stylesheet":"soapMethod"
     },
     soapMethod);
```

Aus dem Flowscript heraus wird also der unten angegebene Matcher `soapAsText` in der Sitemap aufgerufen. Als Parameter wird der zu transformierende Inhalt und der Name des anzuwendenden Stylesheets übergeben. Hierbei gilt die Konvention `"Variablenname":variablenWert`.

```
<map:match pattern="soapAsText">
    <map:generate type="jx" src="xml/inject.jx">
        <map:parameter
            name="soapData"
            value="{flow-attribute:soapData}"/>
    </map:generate>
    <map:transform src="xsl/{flow-attribute:stylesheet}.xsl"/>
    <map:serialize type="text"/>
</map:match>
```

In der Sitemap wird nun der erhaltene Inhalt (soapData) mit Hilfe eines JXTemplates generiert. Diese Technik ist aus Kapitel 3, Abschnitt 3.4.3 bekannt, muss an dieser Stelle allerdings erweitert wreden. Auf dies wird später genauer eingegangen. Zunächst soll angenommen werden, dass der aus dem Flowscript übergebene Inhalt als SAX-Event-Strom für die nun folgende Transformation generiert worden ist und hier zur Verfügung steht.

Mit dem folgenden Stylesheet kann nun der Name der Methode, dessen Position bekannt ist, als Text ausgegeben werden:

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>

    <xsl:template
        match="/soap/soapenv:Envelope/soapenv:Body/*[1]">
        <xsl:value-of select="local-name(.)"/>
    </xsl:template>

</xsl:stylesheet>
```

In dem Template dieses Stylesheets wird auf den ersten Knoten nach dem `<Body>`-Element „gematched“ und dann der Name des so gefundenen

Knotens ausgegeben. Diese Ausgabe der Transformation wird schließlich als Text in den übergebenen Stream serialisiert und steht von nun an im Flowscript zur weiteren Verarbeitung zur Verfügung.

Nachdem bekannt ist, welche Methode der Klient aufgerufen hat, müssen nun die eventuellen Parameter generiert werden. Die Anzahl und Typen der Parameter sind durch die Schnittstellendefinitionen bekannt und im WSDL-Dokument abgelegt. Zu diesem Zweck wird der Name der aufgerufenen Methode mit den Namen der existierenden Methoden verglichen, und im Falle eines unbekannten Methodennames wird dem Klienten in der SOAP-Antwort mitgeteilt, dass diese Methode nicht unterstützt wird:

```
//setting the answer
var ret="The method you have called " +
        "is not understood by this server. Sorry!";
cocoon.sendPage("answer", {"ret":ret});
return;
```

Hierbei wird der Mechanismus benutzt, der in Abschnitt 4.2.6 geschildert ist.

4.2.4 Generierung der Parameter

Im Beispiel der Registrierung eines Zitier-DOIs, wurde also die Methode `registerCitationDOI` aufgerufen, welche als Parameter den Inhalt der Metadatendatei als String und einen weiteren String mit der zum DOI gehörenden URL erwartet. Diese beiden Parameter werden nun mit Hilfe des generischen Matchers `soapAsText` generiert und in den Variablen `soapXML` und `soapURL` gespeichert. Dieser Teil des Flowscripts stellt sich somit folgendermaßen dar:

```
1  if(soapMethod == "registerCitationDOI"){
2      //generating the parameters for
3      //'registerCitationDOI(String xml, String url)'
4      var soapXML = new java.io.ByteArrayOutputStream();
5      cocoon.processPipelineTo(
6          "soapAsText",
7          {"soapData":soapData,
8              "stylesheet":"soapRegisterCitationDOI_XML"
9          },
10         soapXML);
11
12     var soapURL = new java.io.ByteArrayOutputStream();
13     cocoon.processPipelineTo(
14         "soapAsText",
15         {"soapData":soapData,
16             "stylesheet":"soapRegisterCitationDOI_URL"
17         },
18         soapURL);
19 ...
```

In Zeile 1 wird der Name der Methode geprüft und anschließend die Variable, die später die gesendeten XML-Metadaten enthalten wird, `soapXML` instanziiert. Nun wird nach dem bekannten Prinzip der Matcher `soapAsText` aufgerufen, dem wieder der Inhalt des SOAP-Envelopes übergeben wird. Diesesmal soll allerdings das Stylesheet `soapRegisterCitationDOI_XML.xsl` für die Transformation benutzt werden. Das Ergebnis der Transformation wird in der Variablen `soapXML` gespeichert. Mit der gleichen Technik und dem Stylesheet `soapRegisterCitationDOI_URL.xsl` wird anschließend die Variable `soapURL`, die als zweites Argument im SOAP-Envelope enthalten war, mit Inhalt gefüllt.

Die verwendeten Stylesheets unterscheiden sich nur in der Auswahl des Argumentes in Zeile 10. Das Ergebnis eines `'diff'`s lautet deshalb:

```
10c10
<         <xsl:value-of select="ns1:arg0"/>
---
>         <xsl:value-of select="ns1:arg1"/>
```

Prinzipiell sind die verwendeten Stylesheets folgendermaßen aufgebaut:

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="CodataWS">

    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>

    <xsl:template match="/soap/soapenv:Envelope/
soapenv:Body/ns1:registerCitationDOI">
        <xsl:value-of select="ns1:arg0"/>
    </xsl:template>
</xsl:stylesheet>
```

Für die weitere Verarbeitung fehlt nun noch der DOI, der registriert werden soll. Dieser steht in den Metadaten, die in der Variablen `soapXML` gespeichert sind und kann wie folgt generiert werden:

```
var soapDOI = new java.io.ByteArrayOutputStream();
cocoon.processPipelineTo(
    "soapAsText",
    {"soapData":soapXML,
        "stylesheet":"soapDOIFromXML"
    },
    soapDOI);
```

Hierbei wird nun also, wieder unter Verwendung des bekannten Matchers `soapAsText`, eine Transformation durchgeführt, bei der diesesmal allerdings als Eingangsdaten der Inhalt der Variablen `soapXML` verwendet wird. Das

verwendete Stylesheet hat hier die Aufgabe, den DOI aus den Metadaten zu extrahieren:

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="CodataWS">

    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>

    <xsl:template match="/soap/resource">
        <xsl:value-of select="DOI"/>
    </xsl:template>

</xsl:stylesheet>
```

Nachdem nun auch der DOI generiert ist, wird dieser auf das richtige Präfix geprüft und es kann mit der Verarbeitung der Methode begonnen werden.

4.2.5 Verarbeitung der Methode

Der Anwendungsfall der Regstrierung eines Zitier-DOIs (Abb. 2.2, S. 15) sieht vor,

- die gesendeten Metadaten zu archivieren,
- eine E-Mail mit der URN und dem DOI in einem speziellen Anhang zu versenden,
- den DOI mit der zugehörigen URL im Handlesystem^{TM} zu registrieren und
- eine aus den Metadaten generierte PICA-Datei auf einem FTP-Server zu speichern.

Zu diesem Zweck wird nun zunächst die URN nach dem in Abschnitt 2.2.2 (S. 10) beschriebenen Verfahren generiert. Anschließend wird der Inhalt für die PICA-DAtei generiert:

```
//generating pica
var fileContent = new java.io.ByteArrayOutputStream();
var date = codataHelper.timeStamp("dd-MM-yy");
cocoon.processPipelineTo(
  "soap2pica",
  {
    "soapData":soapXML,
    "urn":soapURN,
    "date":date
  },
  fileContent);
```

Dies geschieht mit Hilfe eines weiteren Matchers, dem `soap2Pica`-Matcher. Diesem wird neben dem Inhalt der Metadaten noch die URN und eine Zeichenkette, die das aktuelle Datum beschreibt, mitgegeben. Die verwendete Funktion `timeStamp` liefert einen Zeitstempel und ist in der Java-Klasse `codataHelper` implementiert. Diese Klasse stammt aus einer für COcoonDATA entwickelten Bibliothek, die verschiedene Funktionalitäten, wie etwa den E-Mail-Versand oder das Speichern auf einem FTP-Server zur Verfügung stellt. Der zugehörige Matcher in der Sitemap sieht folgendermaßen aus:

```
<map:match
    pattern="soap2pica">
    <map:generate
        type="jx"
        src="xml/inject.jx">
        <map:parameter
            name="soapData"
            value="{flow-attribute:soapData}"/>
    </map:generate>
    <map:transform src="xsl/soapgetArg.xsl"/>
    <map:transform src="xsl/xml2pica.xsl">
        <map:parameter
            name="urn"
            value="{flow-attribute:urn}"/>
        <map:parameter
            name="date"
            value="{flow-attribute:date}"/>
    </map:transform>
    <map:serialize type="text" />
</map:match>
```

Hier wird also zunächst wieder der Inhalt Variablen `soapData` mit Hilfe des noch zu beschreibenden JXTemplates generiert. Dies hat als Nebeneffekt, dass die XML-Beschreibung der Metadaten durch ein zusätzliches, aus dem JXTemplate stammendes, `<soap>`-Tag umschlossen werden. Dieses kann mit dem Stylesheet `xsl/soapgetArg.xsl` entfernt werden und das Ergebnis dieser Transformation kann direkt als Eingabe der nächste Transformation mit dem Stylesheet `xml2pica.xsl` genutzt werden.

Der Transformation der Metadaten in das PICA-Format werden die Parameter `urn` und `date` mitgegeben, die aus dem Flowscript übergeben worden sind. Diese werden dann in der Transformation an der geeigneten Stelle eingesetzt. So wird der URN beispielsweise mit der Anweisung

$$4083 < 1 > html = G < xsl : value - of select = "\$urn" / >$$

in die PICA-Datei und das Datum mit

$$7001 < xsl : value - of select = "\$date" / >: z$$

eingesetzt.

Anschließend werden im Flowscript die vier nötigen Funktionen zum Speichern, Registrieren und für den E-Mailversand aufgerufen:

```
//executing task for the 'registerCitationDOI'-task
var retXML = storeXML(soapXML);
var retURN = sendMailURN(soapURL, doi);
var retDOI = codataHelper.registerDataDOI(soapDOI, soapURL);

var filename= "coData_"+timeStamp()+".pica";
var retPICA = codataHelper.storeFTP(
            ftpserver,
            ftpuser,
            ftppass,
            ftpdir,
            filename,
            fileContent);
```

Dies sind insbesondere die folgenden Methoden:

storeXML erzeugt zunächst einen Dateinamen inklusive einer konfigurierbaren Pfadangabe. Der Dateiname setzt sich dabei aus dem Präfix `coData_`, einem Zeitstempel und dem Suffix `.xml` zusammen. Der an die Flowscript-Methode übergebene Dateiinhalt wird dann zusammmen mit dem Dateinamen inklusive der Pfadangabe im Aufruf der `storeXML`-Methode der `coDataHelper`-Klasse übergeben. Diese erzeugt dann die Datei und speichert sie an der angegebenen Stelle.

sendMailURN generiert zunächst eine XML-Beschreibung für Registrierung des URNs und versendet diesen anschließend.

registerDataDOI aus der Hilfsbibliothek wird direkt aufgerufen und es werden der DOI und der URN übergeben. Die Methode der `soDataHelper`-Klasse registriert dann den DOI im Handlesystem.

storeFTP ebenfalls aus der Hilfsbibliothek wird direkt aufgerufen und der bereits generierte PICA-Dateiinhalt wird auf dem im Konfigurationsteil des Flowscriptes eingestellten FTP-Server gespeichert.

Nun sind alle Aufgaben, wie sie für diesen Anwendungsfall vorgesehen sind ausgeführt und es wird eine Antwort für den Klienten generiert.

4.2.6 Senden der Antwort

Jede der vier benutzten Methoden liefert einen Rückgabewert vom Typ String. Dieser Rückgabewert wird in den Variablen `retXML, retURN, retDOI` und `retPICA` gespeichert und dient als Benachrichtigungsmöglichkeit für den Klienten. Für die Antwort an den Klienten werden die Nachrichten

der einzelnen Verarbeitungsschritte in einer Zeichenkete zusammengefasst und gesendet:

```
//shipping the answer back to the customer
var ret = "\nstoreXML    : " + retXML + "\n" +
          "sendURN     : " + retURN + "\n" +
          "registerDOI: " + retDOI + "\n" +
          "storePica   : " + retPICA +"\n";
cocoon.sendPage("answer", {"ret":ret});
codataHelper = null;
return;
```

Mit der Funktion `sendPage([String] uri, [Object] bean)` wird normalerweise eine Antwortseite an den Klienten gesendet. In diesem Fall ist diese Antwortseite nicht in HTML, sondern in XML geschrieben und stellt eine SOAP-Response dar. Der zugehörige Matcher `answer` generiert eine „Dummy“-Datei und erstellt durch eine Transformation einen Antwortumschlag, der schließlich als XML an den Klienten gesendet wird:

```
<!-- sending the answer to the client-->
<map:match pattern="answer">
    <map:generate src="xml/dummy.xml" />
    <map:transform src="xsl/soapAnswer.xsl">
        <map:parameter name="ret" value="{flow-attribute:ret}"/>
    </map:transform>
    <map:serialize type="xml"/>
</map:match>
```

Hierbei wird die Antwortnachricht als Parameter für die Transformation aus dem Flowscript übergeben und durch das Stylesheet in den SOAP-Envelope eingefügt:

```
<?xml version="1.0"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="ret"/>

<xsl:template match="/">

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:helloResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/"
      xmlns:ns1="CodataWS">
      <ns1:helloReturn xsi:type="xsd:string">
        <xsl:value-of select="$ret"/>
      </ns1:helloReturn>
    </ns1:helloResponse>
  </soapenv:Body>
```

```
</soapenv:Envelope>

</xsl:template>
</xsl:stylesheet>
```

4.2.7 JXTemplate

Für das hier verwendete JXTemplate wird im Flowscript eine Methode vorgehalten, die aus einem gegebenen String einen SAX-Event-Strom generieren kann. Diese Methode wird als Sessionattribute durch

```
//enabeling the jxtemplate
cocoon.session.setAttribute( "saxer", stringToSAX  );
```

gesetzt und ermöglicht es innerhalb des JXTemplates die folgende Methode aufzurufen, um den String zu parsen und als SAX-Event-Strom in der Pipeline zur weiteren Verarbeitung Verfügung zu stellen.

```
/**
 * function to generate SAX from String
 */
function stringToSAX( str, consumer, ignoreRootElement )
{
  var is = new Packages.org.xml.sax.InputSource(
    new java.io.StringReader( str ) );
  var ignore = ( ignoreRootElement == "true" );
  var parser = null;
  var includeConsumer =
      new org.apache.cocoon.xml.IncludeXMLConsumer(
          consumer, consumer );
  includeConsumer.setIgnoreRootElement( ignore );
  try {
      parser = cocoon.getComponent(
          Packages.org.apache.excalibur.xml.sax.SAXParser.ROLE);
      parser.parse( is, includeConsumer );
  } finally {
      if ( parser != null ) cocoon.releaseComponent( parser );
  }
}
```

Im JXTemplate selber wird ein Makro definiert, welches diese Methode aufruft:

```
<?xml version="1.0"?>
<soap xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
  <jx:macro name="inject">
  <jx:parameter name="value"/>
  <jx:parameter name="ignoreRootElement" default="false"/>
  <jx:set
    var="ignored"
    value="${cocoon.session.saxer(
       value,
       cocoon.consumer,
       ignoreRootElement
```

```
        )}"/>
 </jx:macro>
 <inject value="${soapData}"
                ignoreRootElement="false"/>
</soap>
```

Der prinzipielle Aufbau ist in einem Sequenzdiagramm verdeutlicht (Abb. 4.2, S. 70).

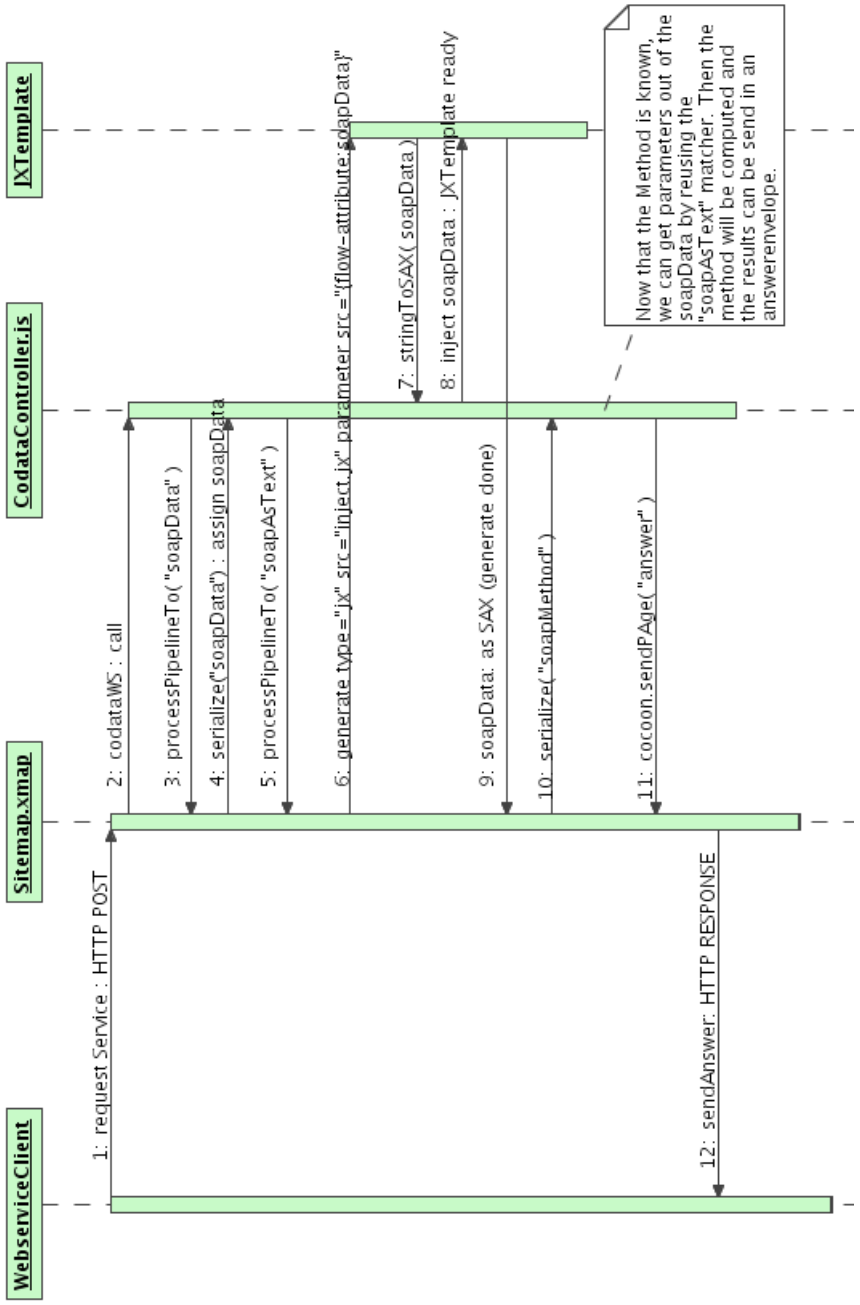

Abbildung 4.2: Sequenzdiagramm (Webinterface und Web Service Klient)

4.3 Erweiterbarkeit

Nachdem nun der Web Service beschrieben ist, stellt sich die Frage, wie man eventuelle Erweiterungen einbringen kann. Der Registrierungsdienst besteht im Wesentlichen aus den Dateien

- sitemap.xmap,
- coDataController.js,
- einer Reihe von Stylesheets und
- der Bibliothek `codata,jar`.

Soll beispielsweise eine neue Methode hinzugefügt werden, die einen Zeitstempel in gewünschter Formatierung liefert, muss zunächst die If-Kaskade im Flowscript um eine `if`-Anweisung über den neuen Methodennamen erweitert werden. Hierbei wird zunächst der Methodenname abgefragt und anschließend müssen die Parameter aus der Anfrage generiert werden. Ist dies erfolgt, können die nötigen Verarbeitungsschritte durchgeführt werden, indem beispielsweise Methoden aus der `codata`-Bibliothek aufgerufen werden. Schließlich muss eine Antwort an den Klienten gesendet werden.

Eine Erweiterung des Flowscriptes um eine Methode, die einen Zeitstempel als Antwort liefert, könnte beispielsweise folgendermaßen aussehen:

```
...
} else if (soapMethod == "timestamp"){

     //generate the parameter
    var soapParameter = new java.io.ByteArrayOutputStream();
    cocoon.processPipelineTo(
        "soapAsText",
        {
            "soapData":soapData,
            "stylesheet":"soapTimestampFormat"
        },
        soapParameter);

    //process the method
    var answer = codataHelper.timeStamp(soapParameter);

    //send the answer
    cocoon.sendPage("answer", {"ret":answer});

    return;
}
```

Hierbei übergibt der Klient einen String zur Formatierung des Zeitstempels. Sendet er beispielsweise die Zeichenkette `"dd-MM-yyyy"` erhält er als Antwort ein Datum mit einer zweistelligen Zahl, die den Tag repräsentiert,

gefolgt von einem Bindestrich, gefolgt von einer zweistelligen Zahl, die den Monat repräsentiert, gefolgt von einem Bindestrich und schließlich die vierstellige Jahreszahl.

Nachdem das Flowscript angepasst ist, muss nun noch das benutzte Stylesheet `soapTimestampFormat.xsl` erzeugt werden, welches dafür verantwortlich ist, den im SOAP-Envelope enthaltenen Parameter in das Flowscript zurückzugeben:

stsetlanguage=XML

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="CodataWS">

    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>

    <xsl:template
        match="/soap/soapenv:Envelope/soapenv:Body/ns1:timestamp">
        <xsl:value-of select="ns1:arg0"/>
    </xsl:template>

</xsl:stylesheet>
```

Bei kompilzierteren Methoden kann es zusätzlich nötig sein, weitere Transformationen durchzuführen. In diesem Fall kann die Sitemap um weitere spezialisierte Matcher erweitert werden.

Die Web Service-Schnittstelle lässt sich also relativ leicht anpassen bzw. erweitern. Aber auch die Browserschnittstelle lässt sich an verschiedene neuen Szenarien anpassen. Hierzu kann der Inhalt der XML-Dokumente für das Formular und gegebenenfalls auch das zugehörige Stylesheet angepasst werden, sodass klientenspezifische Aspekte berücksichtigt werden können. Es kann also ohne großen Aufwand ein Interface geschaffen werden, bei dem beispielsweise die Metadaten direkt eingegeben und anschließend in das PICA-Format umgewandelt werden können. Auch ist man nicht abhängig vom PICA-Format. Tauscht man das Stylesheet, welches für die Transformation der Metadaten verantwortlich ist, aus, ist auch jedes andere Format denkbar. Hierbei muss dann nur ein entsprechendes Stylesheet erstellt werden.

Weiterhin ist es auf Grund der Java-Tachnologie möglich, ohne großen Aufwand die gelieferten Metadaten in eriner Datenbank persistent zu machen. Hierfür müsste beispielsweise nur die Methode aus der `codata`-Bibliothek, die für die Speicherung der Metadaten zuständig ist, entsprechend angepasst werden, sodass die Daten nicht im Dateisystem sondern in einer Datenbank ablegt werden.

Kapitel 5

Performanztest

Um einen Vergleich der alten und der neuen Version zu erhalten, wurde eine Performanzmessung durchgeführt. Hierbei wurden 4000 Daten-DOIs mit der alten Schnittstelle und 4000 mit der neuen registriert.

5.1 Messung

Für die Messung wurde bei jeder Registrierung der Anfangs- und Endzeitpunkt voneinander abgezogen, so die Dauer einer Registrierung ermittelt und in einer Log-Datei festgehalten. Mit Hilfe der UNIX-Tools `awk` und `grep` wurden zunächst die relevanten Zeilen aus den log-Dateien gefiltert und anschließend die Millisekunden in einer `*.csv` Datei gespeichert. Diese Datei wurde anschließend in eine Tabellenkalkulationsprogramm eingelesen und entsprechend aufbereitet.
Der wesentliche Zeitanteil einer Registrierung im Handlesystem liegt allerdings in der Ausführung von Fremd-Code, der nicht verändert werden kann. Deshalb wurde keine wesentliche Performanzsteigerung erwartet und es ist ein gutes Ergebnis, wenn es zu keinen Zeiteinbußen durch den Umbau kommt.

5.2 Resultat

Durch den Umbau kam es, wie erwartet, zu keinen signifikanten Performanzänderungen. Dies ist ein gutes Ergebnis, da auf Standardtechnologie verzichtet wurde. Es wurde sogar eine minimale Geschwindigkeitssteigerung (ca. 70 ms) erreicht. In Abbildung 5.1 (S. 74) ist das Ergebnis zusammengefasst. Rechnet man nun noch den Anteil heraus, für den der Fremdcode verantwortlich ist, ergeben sich die Werte, die in Abbildung 5.2 (S. 74) dargestellt sind.

Die Messung hat eine geringe Abweichung ergeben, was auf relativ genaue Werte hindeutet und die Beobachtung einer Performanzsteigerung stärkt.

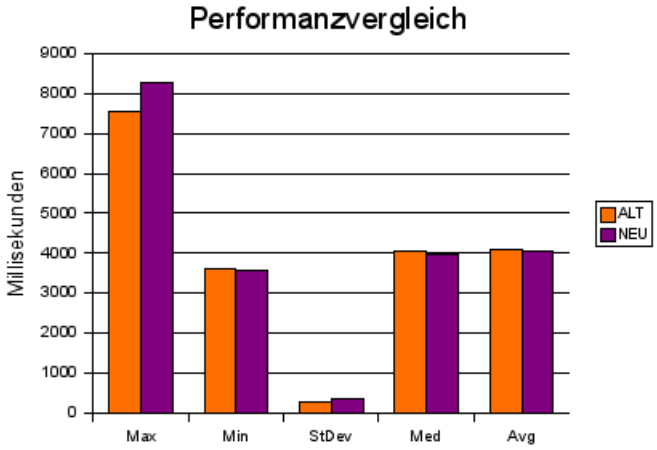

Abbildung 5.1: Resultat der Geschwindigkeitsmessung

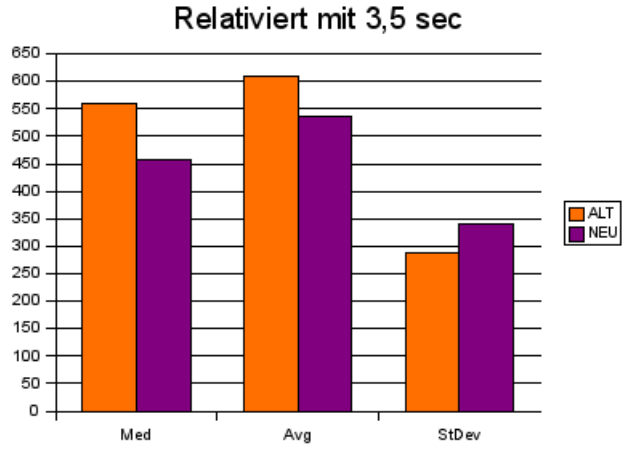

Abbildung 5.2: Performanzvergleich nach Relativierung

Kapitel 6

Zusammenfassung

In diesem Kapitel werden die bisher gewonnenen Informationen zusammengefasst und ein Ausblick in die zukünftige Entwicklung des Projektes gegeben.

6.1 Bewertung und Status

Im Rahmen dieser Bachelorarbeit ist es gelungen, mit dem Web Development Framework Cocoon einen Web Service für die Registrierung von Primärdaten zu erstellen.

Das Programm verfügt nun über zwei Schnittstellen, das Browserinterface und die neu geschaffene Web Service-Schnittstelle. Beide Schnittstellen bieten das gesamte Leistungspektrum von derzeit fünf Methoden an. Die Schnittstellen sind mittels reiner Cocoon-Technologie verwirklicht worden und die bis dahin benötigte Axis-Komponente ist somit obsolet. Dadurch wurden die Probleme, die aufgrund des doppelten Codes gegeben waren beseitigt und die Wartbarkeit signifikant erhöht. Aus dem gleichen Grund wurde die Fehleranfälligkeit bei Erweiterungen wesentlich verringert und es ist relativ einfach, neue Methoden hinzuzufügen. Außerdem konnte die Performanz leicht erhöht werden. Die Anbindung neuer Disziplinen stellt prinzipiell kein Problem dar und durch Anpassung einzelner Stylesheets kann gezielt und einfach auf Kundenwünsche eingegangen werden.

Nachdem der Web Service realisiert worden ist, wurde die zugrundeliegende Idee im Cocoon -Wiki vorgestellt (http://wiki.apache.org/cocoon/WebServiceServer). Auch der Mechanismus, der es erlaubt, Variablen aus dem Flowscript heraus in eine Transformation zu leiten, war scheinbar unbekannt und wurde ebenfalls im Wiki veröffentlicht (http://wiki.apache.org/cocoon/Flow (Beispiel

unten)). Im IRC (irc.freenode.net#cocoon) wurde die Lösung von einigen Mitgliedern der COCOON -Gemeinschaft diskutiert und es gab durchweg positive Resonanz.

Die Installation wurde durch die Registrierung von insgesamt 175.000 DOIs ausgiebig getestet und hat in allen Fällen zuverlässig funktioniert. Außerdem wurden alle fehlenden Methoden implementiert, sodass nun alle definierten Anforderungen umgesetzt worden sind.

6.2 Ausblick

Im Anschluß an diese Bachelorarbeit können noch Verbesserungen und Erweiterungen vorgenommen werden:

Vorschalten eines Apache-Webservers Der Servlet-Kontainer Tomact wird in Zukunft mittles des J2K-Connectors mit einem Webserver Apache verbunden, der von Aussen angesprochen wird. Dies stellt die gängige Praxis dar.

Datenbankpersistenz für die Metadaten Die hochgeladenen Metadaten (XML-Dateien) werden momentan an einer konfigurierbaren Stelle auf der Festplatte des Servers gespeichert. Diese Archivierung wird in Zukunft durch eine Datenbankanbindung abgelöst. Die Realisierung stellt keine größeren Probleme dar, da bereits eine mySQL^{TM}-Instanz vohanden ist. Es wird wahrscheinlich lediglich eine neue Tabelle angelegt, in der die Metadatendatei mit ihrer zugehörigen DOI persistent abgespeichert wird.

Performanzsteigerung im Handlesystem Wegen den langen Laufzeiten bei der Registrierung im Handlesystem steht der Autor in Kontakt mit der Herstellerfirma [6], welche bereits eine neue Version ihrer Software angekündigt hat.

Refaktorisierung der Schnittstellen Die beiden realisierten Schnittstellen mit der darunterliegenden Funktions-Bibliothek können noch stärker entkoppelt werden. So wird eine noch höhere Granularität erreicht.

Momentan wird für jeden Parameter einer Web Service-Methode ein eigenes Stylesheet erstellt. Der einzige Unterschied in diesen Stylesheets ist jedoch der XPath-Ausdruck zur Selektion der Parameter, sowie der Name des Parameters selbst. Außderdem kann in künftigen Szenarien der Ausgabetyp des Stylesheets variieren. Hier wäre es sinnvoll einen Mechanismus zu schaffen, der ein generisches Stylesheet bereithält, dem die entsprechenden Daten als Parameter mitgegeben werden können.

Abbildungsverzeichnis

2.1 Anwendungsfalldiagramm (Webinterface und Web Service Klient) 12
2.2 Aktivitätsdiagramm (Registrierung eines ZitierDOIs) 15
2.3 Aktivitätsdiagramm (Registrierung eines DatenDOIs) 15
2.4 Aktivitätsdiagramm (Aktualisierung einer URL) 16
2.5 Aktivitätsdiagramm (Aktualisierung von Metadaten) 16
2.6 Aktivitätsdiagramm (Daten-DOI zu Zitier-DOI) 17
2.7 Das Formular des Browserinterfaces 19
2.8 Sequenzdiagramm (Webinterface und Web Service Klient) . . 22

3.1 Architektur von CoData 24
3.2 Zertifikat des Dienstes zur Registrierung von Primärdaten . . 26
3.3 Eine simple XML-Datei 28
3.4 Prinzipieller Ablauf einer Transformation 30
3.5 Cocoons Architektur des *Separation of Concerns* 36
3.6 Die durch Cocoon produzierte Ausgabe 42
3.7 Flowscript des Taschenrechners 43
3.8 Eine skeletthafte SOAP-Nachricht 51

4.1 Anwendungsfalldiagramm (Webinterface und Web Service Client) 56
4.2 Sequenzdiagramm (Webinterface und Web Service Klient) . . 70

5.1 Resultat der Geschwindigkeitsmessung 74
5.2 Performanzvergleich nach Relativierung 74

Literaturverzeichnis

[1] Forschungszentrum L3S
http://l3s.de

[2] TIB hannover
http://www.tib.uni-hannover.de/

[3] Deutsche Forschungs Gemeinschaft
http://www.dfg.de

[4] Gemeinsamer Bibliotheksverbund
http://gbv.de

[5] The International DOI Foundation
http://www.doi.org/

[6] The Corporation for National Research Initiatives
http://cnri.reston.va.us/

[7] The Handlesystem developed by the CNRI http://www.handle.net/
Handle System Protocol Specification:
http://www.handle.net/rfc/rfc3652.html

[8] WDC Mare
World Data Center for Marine Environmental Sciences
http://www.wdc-mare.org/

[9] The Apache Software Foundation
http://httpd.apache.org/

[10] Apache Jakarta Tomcat
http://jakarta.apache.org/tomcat/index.html

[11] The Apache Cocoon Project
http://cocoon.apache.org/

[12] The Apache Ant Project
Apache Ant is a Java-based build tool. In theory, it is kind of like Make, but without Make's wrinkles. http://ant.apache.org/

[13] The Apache ¡Web Services/¿ Project
http://ws.apache.org/axis/

[14] W3C XML Protocol Working Group
http://www.w3.org/2000/xp/Group/

[15] Persistent Identifier Management an Der Deutschen Bibliothek
http://www.persistent-identifier.de/

[16] Java Technology
http://java.sun.com

[17] W3C Extensible Markup Language
http://www.w3.org/XML/

[W3C] W3C. Web Services Architecture Requirements, Oct. 2002.
http://www.w3c.org/TR/wsa-reqs.

Anhang A

Programm-CD

Auf der beiligenden CD-ROM befindet sich die momentane Programmversion mit dem Namen `cocoon.2005.05.05.tar.gz`. Das Programm kann durch entpacken in den `webapps`-Ordner einer Tomcat-Installation in Betrieb genommen werden. Nähere Informationen sind in den Dateien `INSTALL.txt` und `README.txt` im Release ethalten.

Ferner ist auf der CD-ROM der Quelltext der Hilfsbibliothek enthalten. Für die Kompilation existiert ein Ant-Skript, welches an die lokale Installation angepasst werden muss. Das Ant-Skript geht von einer Tomcat-Installation mit ebenfalls installiertem COcoonDATA aus. Es kompiliert dann die Quelltexte und erstellt eine `codata.jar`-Datei, die anschließend in den `WEB-INF/lib`-Ordner der COcoonDATA-Installation kopiert wird.

Zusätzlich befindet sich auf der CD-ROM eine Beispielimplementation eines Web Service Klienten, der die `hello`-Methode des Web Services aufruft.

Alle sensiblen Daten in den entsprechenden Dateien wurden aus datenschutzrechtlichen Gründen durch `'*'` ersetzt.

Anhang B

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 9. Mai 2005

_____

Eingegangen am (Datum/Stempel):_____

Anhang C

Danksagung

Bedanken möchte ich mich bei den folgenden Personen, die alle dazu beigetragen haben, dass ich diese Bachelorarbeit und somit den Abschluss meines Bachelorstudiums erreichen konnte:

Herrn Prof. Dr. Wolfgang Neidl danke ich, dass er es mir ermöglicht hat, dieses Thema zu bearbeiten.
Ebenso Frau Prof. Dr. Nicola Henze, die mir durch ihre Vorlesung *Semantic Web* viele Anregungen gegeben hat.
Bei meinem Betreuer Herrn Dr. Jan Brase, der mir immer mit gutem Rat und Ideen zur Seite stand, möchte ich mich besonders bedanken.

Weiterhin danke ich Herrn Prof. Dr. Rainer Parchmann, der mich in meinem Entschluss, den Studienwechsel vom Studium der Medizin zur Informatik zu vollziehen, bestärkt hat. Gleichfalls gilt der Dank Claus Lahner, der mir von dem damals neuen Studiengang 'Angewandte Informatik' erzählt hat.
Auch bei Frau Franziska Pfeffer möchte ich mich für den Kaffee und die netten Gespräche bedanken.
Rolf Kulemann sowie der COCOON-Community gilt ebenso besonderer Dank, für die Anregungen und Diskussionen zu den verschiedenen Lösungswegen.
Ganz besonders bedanke ich mich bei meinem Sohn Nick Elias Kirsch, einfach weil er da ist, sowie meiner Lebensgefährtin Katharina Kirsch die mir in schwierigen Situationen zur Seite steht.
Schließlich bedanke ich mich bei meinen Eltern Helga und Dieter Hinzmann und meiner Schwester Meike, für die tolle Unterstützung während des Studiums und dem Korrekturlesen dieser Arbeit.