Konzeption und Implementierung einer verteilten Institutionsverwaltung als anwendungsspezifische Form eines verteilten Metadaten-Repository

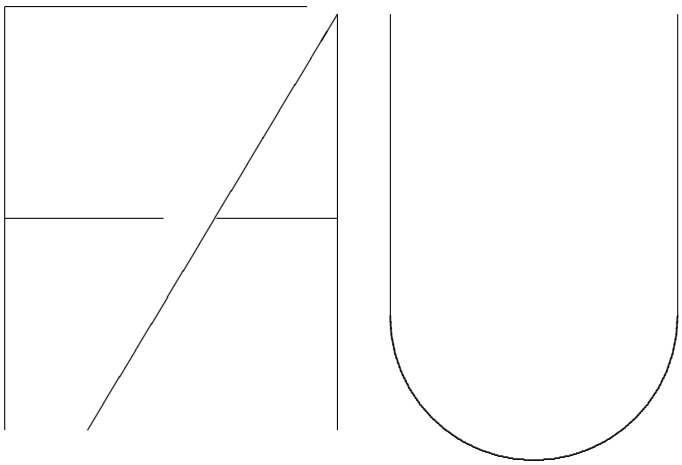
Studienarbeit

Igor Engel

Lehrstuhl für Informatik 6
(Datenmanagement)

Department Informatik
Technische Fakultät

Friedrich Alexander-
Universität
Erlangen-Nürnberg



Konzeption und Implementierung einer verteilten Institutionsverwaltung als anwendungsspezifische Form eines verteilten Metadaten-Repository

Studienarbeit im Fach Informatik

vorgelegt von

Igor Engel

geb. 18.08.1980 in Aktjubinsk

angefertigt am

Department Informatik
Lehrstuhl für Informatik 6 (Datenmanagement)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Univ.-Prof. Dr.-Ing. habil. Richard Lenz
Dipl.-Inf. Christoph P. Neumann

Beginn der Arbeit: 08.03.2010
Abgabe der Arbeit: 04.07.2011

Erklärung zur Selbständigkeit

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Informatik 6 (Datenmanagement), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Studienarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 04.07.2011

(Igor Engel)

Kurzfassung

Konzeption und Implementierung einer verteilten Institutionsverwaltung als anwendungsspezifische Form eines verteilten Metadaten-Repository

Im Gesundheitswesen kooperieren mehrere Institutionen und tauschen die Informationen untereinander aus. Diese Informationen können sowohl Patientendaten als auch die diese Institutionen beschreibenden Informationen sein. Mögliche Institutionen im Gesundheitswesen sind Krankenhäuser oder Facharztpraxen. In dieser Studienarbeit geht es um die verteilte Institutionsverwaltung. Die Verwaltung bezieht sich auf die Informationen, die die Institutionen selbst beschreiben. Dabei sollen die Institutionen autonom bleiben und ihre Informationen selbständig verwalten. Die Kerninformationen sind die Aktoren und die Rollen von den Aktoren in diesen Institutionen. Das Ziel ist die Software, die eine verteilte Institutionsverwaltung erlaubt. Dafür werden einige mögliche Technologien untersucht. Mit den ausgewählten Technologien wird dann die angestrebte Software realisiert. Schließlich wird die Realisierung des Systems auf einige Eigenschaften wie Skalierbarkeit untersucht.

Abstract

Design and implementation of a distributed Institution management as an application-specific form of a distributed metadata repository

In health care many institutions cooperate and share information among themselves. This information can be both patient data and the descriptive information for these institutions. Possible health care institutions are hospitals or specialist clinics. This study is about the distributed organization management. The administration refers to the information describing the institutions themselves. The institutions should be autonomous. They should also manage their own information. The core information is the actors and the roles of the actors in these institutions. The goal is the software that enables a distributed organization management. Some possible technologies are investigated. The desired software is realized then with selected technologies. Finally, the realization of the system on some properties such as scalability is studied.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
OASIS	Organization for the Advancement of Structured Information Standards
RDF	Resource Description Framework
XDI	XRI Data Interchange
XML	Extended Markup Language
XRDS	Extensible Resource Descriptor Sequence
XRD	Extensible Resource Descriptor
XRI	Extensible Resource Identifier
GCS	Global Context Symbols
SEP	Service Endpoint
PDS	Personal Data Store
DBMS	Database Management System
DSSP	DataSpace Support Platform

1 Einleitung

Das Gesundheitssystem ist ein sich rasch entwickelndes Gebiet. Diese schnelle Entwicklung hat Auswirkungen auf allen Ebenen: politischen, sozialen, wirtschaftlichen, medizinischen und technischen. Dabei spielt die enge Kooperation verschiedener Institutionen des Gesundheitswesens eine große Rolle. Die Kooperation und die Koordination stellen hohe Anforderungen an die zu entwickelnden technischen Systeme. Diese Arbeit ist ein Teil des Projekts, das sich mit der Forschung auf dem Gebiet der Zusammenarbeit von Institutionen im Gesundheitswesen befasst.

1.1 Motivation

Das Ziel des übergeordneten Forschungsprojekts „ProMed“ ist die Umsetzung einer verteilten Prozessunterstützung im Bereich der Medizin, namens „alpha-Flow“. Weiterführende Informationen liefern [Len09] und [Chr09]. In verteilten Behandlungsprozessen kooperieren mehrere Institutionen in unterschiedlichen Rollen miteinander. Dabei müssen sie die Informationen untereinander austauschen. Jede Institution muss in der Lage sein die Informationen über sich selbst zu verwalten und bei Bedarf den Partnerinstitutionen bereitzustellen.

1.2 Aufgabenstellung

Die Aufgabe dieser Arbeit ist es durch eine umfassende Recherche einen Überblick über die verfügbaren Ansätze zur Institutionsverwaltung zu eruieren. Im Fokus stehen autonome und lose-gekoppelte Systeme. Ziel ist die Konzeption und prototypische Implementierung einer Software-Komponente in Java, die es erlaubt, Institutionsinformationen zwischen autonomen Systemen abzufragen und zu synchronisieren. Die Idee ist, dass die Institutionen die Daten, die die Institutionen beschreiben, selbständig verwalten.

Dies soll verteilt geschehen. Dabei haben die Institutionen volle Kontrolle über ihre Daten. Die Institutionen sollen außerdem die Möglichkeit bekommen ihre Daten unter-

einander austauschen zu können. Dafür ist eine Infrastruktur notwendig, die aufgebaut werden soll. Eine vielversprechende Technologie dafür ist XRI, XRDS und XDI.

1.3 Methodik

Im ersten Schritt werden die Technologien untersucht, die für die Lösung der Aufgabe in Frage kommen könnten. Dabei werden die Überlegungen gemacht, wie die Institutionen und Aktoren in den Institutionen identifiziert und lokalisiert werden, und wie der Informationsaustausch stattfindet. Außer den genannten Fragen werden diese Technologien auf die Verteilung und Skalierbarkeit untersucht. Im weiteren Verlauf wird durch die verwandten Arbeiten der Überblick über die aktuellen Entwicklungen in den verwandten Gebieten verschaffen. Darunter fallen die Verwaltung von Rollen, die die Aktoren in den Institutionen einnehmen können, und die Möglichkeiten der Verwaltung von eigenen Institutionsinformationen. Danach werden die Anforderungen an das zu entwickelnde System identifiziert. Mit den gewählten Technologien und gefundenen Anforderungen wird im nächsten Schritt der Lösungsansatz ausgearbeitet. Dieser Ansatz liefert die Informationen bezüglich der Identifikation von Instanzen und der Infrastruktur für den Datenaustausch zwischen ihnen. Nachdem der Ansatz zur Lösung der Aufgabe fertig gestellt ist, wird das ganze System entworfen und technisch umgesetzt. Dabei werden einige UML-Diagramme wie Sequenzdiagramme oder CRC-Diagramme und auch Freiformen der besseren Darstellung zum Einsatz gebracht.

2 Grundlagen

Dieses Kapitel liefert eine Übersicht über die Grundlagen von XRI, XRDS, XRD und XDI. Das sind relativ junge Technologien. Diese werden in dieser Arbeit als Schlüsseltechnologien für die Lösung der Aufgabe herangezogen und werden hier bis zu einem bestimmten Detailgrad erläutert. Dabei wurde die Originalbezeichnung „Authority“ ins Deutsche als „Instanz“ übersetzt. Authority wird jedoch an einigen Stellen im Original verwendet.

2.1 XRI

XRI dient der Identifikation von Instanzen. Zunächst werden die Elemente von XRI eingehend aufgezeigt.

- XRI, Extensible Resource Identifier, digitaler Identifizierer für beliebige Ressourcen.
- XRDS, Extensible Resource Descriptor Sequence, XML-Dokumentenformat für die Auffindung und Auflösung von Ressourcen, die durch XRIs identifiziert werden.
- i-name und i-number sind zwei Formen von XRI. i-name ist menschenfreundliche, i-number maschinenfreundliche XRI.
- Authority - die Entität für die Kontrolle des Namensraumes.
- Subsegmente sind Teile von XRI, die durch Trennzeichen „*“ oder „!“ getrennt sind.
- GCS, Global Context Symbols, ein-Zeichen-Symbole, die den Typ einer XRI im globalen Kontext festlegen.
- Eine XRI - ein Segment - besteht aus Subsegmenten, die i-names oder i-numbers sein können und entweder durch „!“ bei i-numbers oder durch „*“ bei i-names getrennt sind. XRIs beginnen mit einem GCS-Symbol, das den Typ (Organisation/Person/Generisch) der XRI angibt.

XRI steht für Extensible Resource Identifier. Das ist der Bezeichner für digitale Identifizierer, die zu der XRI-Syntax-Spezifikation [OAS08] konform sind. XRI ist eine Identifikations- und Auflösungsinfrastruktur sowie DNS - Domain Name System. Als Protokoll für die Auflösung wird bei XRI HTTP(S) verwendet. Dabei wird eine XRI zu einem XRDS (Extensible Resource Descriptor Sequence)-Dokument aufgelöst. Den Server, der in der XRI-Infrastruktur für beliebige XRI-Instanzen die XRDS-Dokumente verwaltet, nennt man Instanzenserver (engl. authority server) [Ree08]. Es gibt zwei Formen von XRI, i-name und i-number. i-name ist eine Form menschenfreundlicher XRI, die übertragbar ist. Sie kann einem neuen Eigentümer übertragen und einer neuen Ressource zugewiesen werden. i-number ist dagegen eine persistente maschinenfreundliche XRI, die nur einmal einer Ressource zugewiesen wird und nicht übertragen werden kann. Das Bild 2.1 als Screenshot liefert ein Beispiel für i-name und i-number. Dabei ist @alpha ein i-name und @!9EAA.2C43.74CF.9A4A (unter Canonical ID) die dazugehörige i-number.

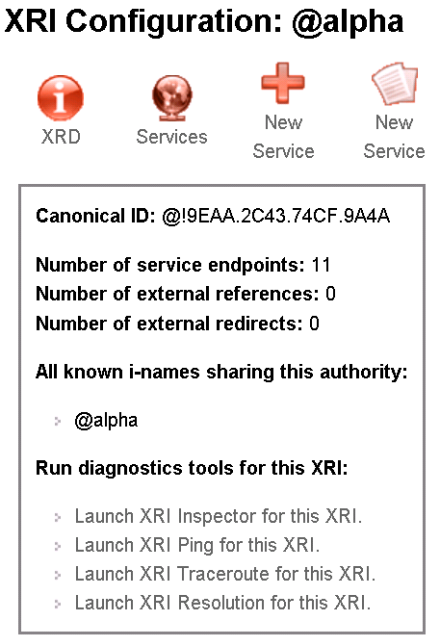


Bild 2.1: XRI: i-name and i-number

So wie bei Domainnamen ist bei i-names und i-numbers die Delegation möglich. Dabei findet die Delegation jedoch von links nach rechts statt und anstatt von Punkten verwendet man für die Delegation bei i-names Sterne und bei i-numbers Ausrufezeichen. XRIs können verschiedene Typen von Ressourcen identifizieren: Personen, Organisationen, Applikationen, digitale Objekte, oder auch abstrakte Konzepte wie „Liebe“ oder „Nummer 21“. XRIs beginnen immer mit einem speziellen Zeichen - GCS. GCS, Global Context Symbols, geben an, um was für eine Art von Instanzen im globalen Kontext es sich

handelt. @-Zeichen wird bei Organisationen verwendet, = bei Personen und + ist ein generisches Kontext-Symbol. Außerdem gibt es noch das Dollar-Zeichen, das für self-context verwendet wird und als Wurzel von der XDI Dictionary dient. Die Teile einer XRI, die bei der Delegation durch Sterne bei i-names und Ausrufezeichen bei i-numbers getrennt sind, nennt man Subsegmente. Dabei ist zu beachten, dass das GCS bereits als erstes Subsegment gilt. Jedes neu angelegte Subsegment führt zu einer separaten Instanz. Bei @alpha existieren zum Beispiel zwei Instanzen, eine für @ und eine für @alpha [OAS08][Dru05].

2.2 XRDS und XRD

XRDS, Extensible Resource Descriptor Sequence, ist ein XML-basiertes Dokumentenformat. Ein XRDS besteht aus mehreren XRDs, mindestens aus einem.

- XML, Extensible Markup Language, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten.
- SEP, Service Endpoint, ist eine Menge von Elementen, die einen Dienst und eine konkrete Netzwerkadresse, unter der dieser Dienst zu erreichen ist, beschreiben. Ein XRD kann dabei mehrere SEPs beinhalten. Das sind Dienste, die mit einer Ressource verknüpft sind und in dem entsprechenden XRD beschrieben sind.
- HTTP, Hypertext Transfer Protocol, ist ein Protokoll zur Übertragung von Daten über ein Netzwerk.

Bei der Suche nach Metadaten einer Ressource wird eine XRI zu XRDS aufgelöst. Die Auflösung ist ein Prozess der Überführung eines Identifizierers in Metadaten, die eine identifizierte Ressource beschreiben. In XRI ist dies als HTTP(S)-Anfrage für ein XRDS-Dokument implementiert. XRDS ist ein XML-basiertes in XRI Resolution 2.0 spezifiziertes Dokumentenformat. XRDS-Dokumente enthalten mehrere XRDs (Extensible Resource Descriptors), mindestens einen. In dem XRDS zu der XRI @alpha*institutions gibt es zum Beispiel drei XRDs, jeweils einen für die Subsegmente @, alpha und institutions. Jeder XRD enthält üblicherweise einen oder mehrere Service Endpoints (SEPs) [Gab08][OAS08]. Ein SEP ist eine Sammlung von Elementen. Diese Elemente beschreiben den Typ des Dienstes, die konkrete Netzwerk-URI, unter der dieser Dienst erreichbar ist und andere Details [OAS08]. Das sind Dienste, die mit einer Ressource verknüpft

sind. Listing 2.1 zeigt solche Dienste, die innerhalb von <Service/>-Elementen beschrieben sind. Außerdem liefert die Erklärung dieses Listings, wie diese Informationen zu interpretieren sind.

Authority Resolution ist die erste Phase von der XRI-Auflösung, bei der die XRI zu einem XRD aufgelöst wird. Dieser XRD beschreibt die Zielinstanz, Instanz die durch das letzte Subsegment in der XRI identifiziert wird. Man kann also eine XRI zu einem XRDS auflösen. Dieser enthält dann die XRDs von allen durch alle Subsegmente einer XRI identifizierte Instanzen. Oder, wie dargestellt, nur zu einem XRD, der die durch das letzte Subsegment identifizierte Instanz beschreibt.

SEP-Auswahl ist die zweite optionale Phase von der XRI-Auflösung. Dabei kann der Auflöser eine spezifische Menge von SEPs auswählen [OAS08].

Hier ist ein Beispiel, Listing 2.1, für die Authority Resolution. Dabei wird bei dem i-name „@alpha*institutions“ der letzte Teil, also „institutions“, zu einem XRD aufgelöst.

```
<?xml version="1.0" encoding="UTF-8"?>
<XRD xmlns="xri://$xrd*($v*2.0)">
 <Query>*institutions</Query>
 <Status cid="verified" code="100">Success</Status>
 <ServerStatus code="100">Success</ServerStatus>
 <Expires>2011-03-20T23:15:32.796Z</Expires>
 <ProviderID>@!9EAA.2C43.74CF.9A4A</ProviderID>
 <LocalID>!ace8.674d.4cec.8393</LocalID>
 <LocalID>*i</LocalID>
 <CanonicalID>@!9EAA.2C43.74CF.9A4A!ace8.674d.4cec.8393</CanonicalID>
 <Service>
  <ProviderID/>
  <Type select="true">xri://$xdi!($v!1)</Type>
  <Path select="true">($context)!($xdi)!($v!1) </Path>
  <MediaType match="default"/>
  <URI append="none">
  http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*institutions/
  </URI>
 </Service>
 <Service>
  <ProviderID>@!9EAA.2C43.74CF.9A4A!ace8.674d.4cec.8393</ProviderID>
  <Type select="true">xri://$res*auth*($v*2.0)</Type>
  <MediaType select="false">application/xrds+xml</MediaType>
  <URI append="none" priority="2">
  http://http://131.188.36.135:80/com.alpha.resolve-0.5/ns/@!9EAA.2C43
     .74CF.9A4A!ace8.674d.4cec.8393/
  </URI>
```

```
27    <URI append="none" priority="1">
28    https://http://131.188.36.135:443/com.alpha.resolve-0.5/ns/@!9EAA.2
          C43.74CF.9A4A!ace8.674d.4cec.8393/
29    </URI>
30   </Service>
31  </XRD>
```

Listing 2.1: XRD von *institutions.

In diesem Beispiel enthält das </CanonicalID>-Element die i-number „@!9EAA.2C43.74CF.9A4A!ace8.674d.4cec.8393“ zu dem i-name „@alpha*institutions“. Beide führen zu der gleichen Instanz. i-number und i-name sind zwei Formen von XRI. Das erste </Service>-Element liefert die Information, dass es sich um den XDI-Dienst handelt. Außerdem enthält es eine konkrete Netzwerkadresse „http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*institutions/“, unter der der XDI-Dienst erreichbar ist. Das zweite </Service>-Element liefert die Informationen, die den Auflösungsdienst betreffen. Die hier hinterlegten Netzwerkadressen sind Adressen, an denen sich die Komponente befindet, die für die Auflösung von Kinder-XRIs von „@alpfa*instituions“ wie zum Beispiel „@alpha*institutions*ucc-asklepios“ verantwortlich ist.

2.3 XDI

XDI(XRI Data Interchange) ist ein Datenmodell und Protokoll zum Tauschen, Verlinken und Synchronisieren von Daten über Netzwerke mittels XML und XRI [OAS04][Dru04]. XDI wird von der OASIS XDI Technical Committee entwickelt. Im Jahr 2007 wurde von dem Committee ein Modell, das auf RDF basiert, entwickelt und heißt XDI RDF Modell. Dieses Modell zusammen mit der dritten Generation von XRI Spezifikation, bestehend aus XRI 3.0 und XRD 1.0, bilden die Basis für die XDI 1.0 Spezifikation. Sowohl XRI 3.0 und XRD 1.0 als auch XDI 1.0 befanden sich im Laufe der Entstehung dieser Arbeit und auch danach in der Entwicklung. Das XDI RDF Modell, das in dieser Arbeit verwendet wird, nutzt bereits die Elemente von XRI 3.0.

Die Grundidee bei XDI ist die Arbeit mit einem XDI-Graphen, der die Struktur eines XDI-Dokumentes darstellt. Ein XDI-Graph besteht möglicherweise aus: Subjekten, Prädikaten, Literalen, Referenzen oder inneren Graphen. Die Daten werden dabei durch XDI-Statements strukturiert und adressiert. Eine Anweisung kann aus vier verschiedenen Typen von XDI-Adressen bestehen: Subjekt-Adresse, Prädikat-Adresse, Objekt-Adresse

oder Context-Adresse. In dieser Arbeit wurden nur drei ersten genannten Typen verwendet.

Im Listing 2.2 wird die Struktur einer Anweisung gezeigt. Da die Struktur und Adressierung gleich sind, kann man die Anweisung auch als XDI-Adresse betrachten. Das xdi-objekt kann dabei ein Literal oder eine Referenz sein.

```
1 xdi-adress = xdi-subject["‘/"’xdi-predicate["‘/"’xdi-object]]
```

Listing 2.2: Struktur der XDI-Anweisung

Das soll an einem Beispiel, Listing 2.3, verdeutlicht werden.

```
1 @alpha*stdcatalogs*rolecatalog/+roles/gynecologist
```

Listing 2.3: Beispiel für eine XDI-Anweisung

Der erste Teil, „@alpha*stdcatalogs*rolecatalog“, ist ein Subjekt. „+roles“ ist ein Prädikat. Und „gynecologist“ ist ein Literal. XDI-Protokol unterstützt vier folgende Operationen: get, add, mod und del. Diese sind äquivalent zu read, create, update und delete. Wenn man beispielsweise etwas einfügen will, etwa ein neues Literal „radiologist“ unter dem Subjekt „@alpha*stdcatalogs*rolecatalog“ und dem Prädikat „+roles“, dann verwendet man die Operation „add“ mit der Anweisung „@alpha*stdcatalogs*rolecatalog/+roles/radiologist“. Danach können die Rollen bzw. die unter dem Prädikat „+roles“ gespeicherte Literale ausgelesen werden. Das macht man mit der Operation „get“ und der Anweisung „@alpha*stdcatalogs*rolecatalog/+roles“. Durch diese Anfrage werden alle Rollen (Literale) zurückgegeben.

Die Struktur zeigt das Beispiel im Listing 2.4.

```
1 @alpha*stdcatalogs*rolecatalog
2     +roles
3         radiologist
4         oncologist
5         gynecologist
```

Listing 2.4: XDI-Struktur.

2.4 Zusammenspiel von XRI, XRDS/XRD und XDI

So wie bereits beschrieben, wird eine XRI zu einem XRDS/XRD Dokument aufgelöst. Aus dem XRD selektiert man dann den gewünschten Dienst, genauer gesagt die Beschreibung des Dienstes, und extrahiert daraus die URI, unter der dieser Dienst zu erreichen ist. XDI

ist ein Beispiel für so einen Dienst. Falls man aus dem bestimmten XRD-Dokument einer bestimmten XRI die Beschreibung des XDI-Dienstes selektiert und die entsprechende URI ausliest, kann man dann die HTTP-Anfragen zu dem unter dieser URI erreichbaren XDI-Dienst (XDI Endpoint genannt) senden und die zu der entsprechenden XRI gehörende Daten abrufen oder auch aktualisieren. Diese Möglichkeit wird bei dieser Arbeit verwendet, um die Institutionen/Aktoren zu verwalten und mit deren Metadaten zu arbeiten. Die beschriebene Verwendung dieser Technologien wird in Form von einem Schichtenmodell im Bild 2.2 dargestellt.

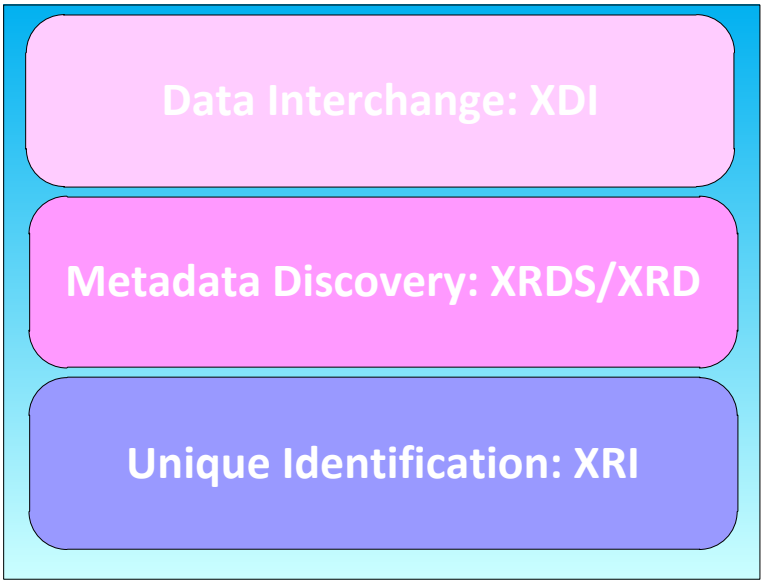


Bild 2.2: Schichtenmodell

Die XRI Resolution ist ein weiteres Beispiel für solche Dienste. Dabei beschreibt das entsprechende Dienstelement des XRD wer für die Subsegmente bei einer Delegation zuständig ist und wo dieser zu erreichen ist. Die Vorgehensweise wird nun an einem Beispiel erklärt.

- Man kennt eine XRI „@alpha*institutions“ und weiß, dass in dem entsprechenden XDI-Dokument zu dieser XRI eine Liste von Institutionen in Form von XRIs gespeichert ist. Man will diese Liste anfordern. Weiterhin weiß man, dass diese Liste im Dokument selbst unter der XDI-Adresse „@alpha*institutions/+institutions“ abgelegt ist. Dies ist im Folgenden, Listing 2.5, dargestellt.

```
1 @alpha*institutions
2     +institutions
3         @alpha*institutions*officeprofalice
```

```
4         @alpha*institutions*officedrbob
5         @alpha*institutions*ucc-asklepios
```

Listing 2.5: Liste von Institutionen innerhalb von XDI.

- Im ersten Schritt muss die XRI zu dem entsprechenden XRD aufgelöst werden. Der entsprechende XRD ist im Kapitel über XRD, Listing 2.1, dargestellt.
- Aus diesem XRD selektiert man nun das Service-Element vom XDI-Dienst (Listing 2.6).

```
1  <Service>
2   <ProviderID/>
3   <Type select="true">xri://$xdi!($v!1)</Type>
4   <Path select="true">($context)!($xdi)!($v!1) </Path>
5   <MediaType match="default"/>
6   <URI append="none">
7   http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*
       institutions/
8   </URI>
9  </Service>
```

Listing 2.6: Service-Element für XDI.

- Und liest anschließend die Netzwerkadresse, unter der der XDI-Dienst läuft. Das ist „http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*institutions/“.
- Im weiteren Verlauf sendet man eine Anfrage an diesen Dienst. Es wird beispielsweise die Liste von Institutionen angefordert. Anschließend bekommt man die Antwort von diesem Dienst und extrahiert daraus die gewünschte Liste (Listing 2.7).

```
1         @alpha*institutions*officeprofalice
2         @alpha*institutions*officedrbob
3         @alpha*institutions*ucc-asklepios
```

Listing 2.7: XRIs von Institutionen.

Das Bild 2.3 zeigt außerdem ein Beispiel der Auflösung einer XRI „@alpha*institutions*ucc-asklepios*idesc“. Im Bild wird sichtbar, dass die einzelnen Subsegmenten entsprechenden Instanzen auf unterschiedlichen Servern verwaltet werden. Jeder Server teilt dem Auflöser mit, welcher Server als nächster für die Auflösung von Subsegmenten verantwortlich ist, bis das letzte Subsegment aufgelöst wird. Dabei kann es immer der gleicher Server sein, der für das nächste Subsegment verantwortlich ist.

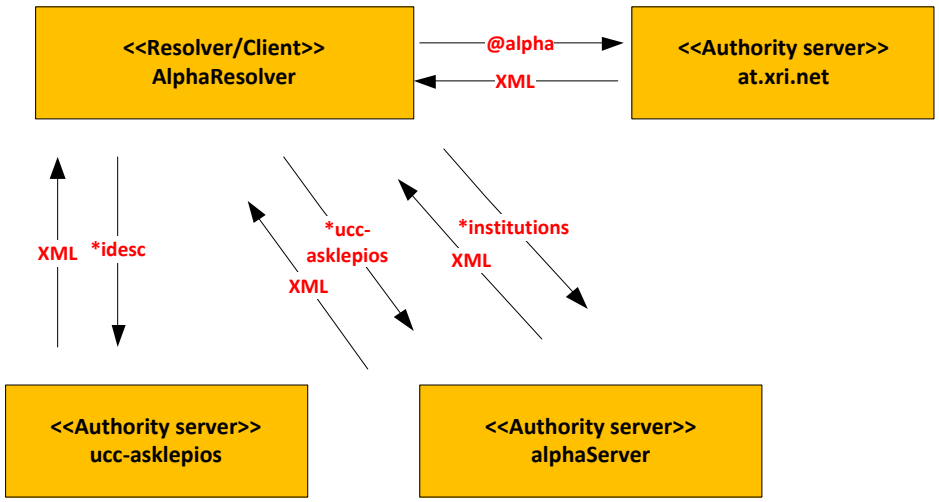


Bild 2.3: Discovery

2.5 Zusammenfassung

In diesem Kapitel werden die Grundlagen vorgestellt. Dazu gehören XRI, XRDS/XRD und XDI. XRI dient also der Verwaltung und Lokalisierung von beliebigen Ressourcen. XRDS/XRD beschreiben die Dienste, die mit diesen Ressourcen verknüpft sind. XDI dient an erster Stelle der Speicherung und dem Tausch von Daten, die zu einer Ressource gehören und diese beschreiben. Diese Möglichkeiten werden bei dieser Arbeit verwendet, um die Institutionen/Aktoren zu verwalten und mit deren Metadaten zu arbeiten. Mit XRI identifiziert man die Institutionen/Aktoren. Mit XDI speichert und tauscht man deren Metadaten aus.

Diese Technologien erlauben uns eine Verteilung und die selbständige Verwaltung der Daten durch Institutionen. Also eine dezentrale Verwaltung. Mit diesen Technologien ist die Realisierung einfacher, da man hier in allen Institutionen gleiche Technologie hat. Das heißt, man hat z. B. keinen Integrationsansatz, den man bei unterschiedlichen Quellen hat, oder keinen verteilten/föderierten Datenbankansatz, bei dem der Aufwand höher ist.

3 Verwandte Arbeiten

In diesem Kapitel werden die Ansätze, die ähnliche bzw. verwandte Themen bearbeiten, kurz erläutert.

3.1 Rollenmodelle

Bei der Suche nach der Lösung für eine Institutionsverwaltung taucht eine wichtige Frage auf. Zu einer Institution gehören Aktoren, die in dieser in unterschiedlichen Rollen tätig sein können. Ein Aktor kann in unterschiedlichen Rollen auftreten. Diese könnten sich bei den jeweiligen Aktoren ändern. Ein Aktor könnte außerdem in unterschiedlichen Institutionen in verschiedenen Rollen auftreten. Die Frage ist also, wie die Rollen von Aktoren realisiert und verwaltet werden sollen. Dazu wurden einige Konzepte von Rollenmodellen untersucht. Solche Konzepte liefern beispielhaft [Tet05] und [Tet02]. Diese werden folgend kurz erläutert.

Mit dem Ziel, die Adoption von Objekten in die Umgebungen zu realisieren, wurde ein rollenbasiertes Modell „Epsilon“ und eine Sprache „EpsilonJ“ vorgeschlagen. In „Epsilon“ ist die Umgebung als das Feld der Zusammenarbeit zwischen Rollen und einem Objekt definiert. Dabei nimmt ein Objekt bei der Adoption in die Umgebung eine Rolle an.

- Objekte repräsentieren Dinge oder Konzepte der realen Welt.
- Objekte in der realen Welt existieren in unterschiedlichen Umgebungen.
- Objekte können durch die Umgebungen wandern.
- Es ist auch möglich, dass sich die Umgebung des Objektes selbst dynamisch ändert.
- Beim Wechsel der Umgebung ändern sich auch die Objekte.
- In einer Umgebung spielt jedes Objekt eine Rolle.
- Objekte können die Umgebungen frei betreten oder diese verlassen, und mehreren Umgebungen gleichzeitig angehören, sodass eine dynamische Anpassung der Objekte realisiert wird.

Die Grundprinzipien von diesem Modell sind:

- Unterstützung der adaptiven Evolution: In „Epsilon“ entstehen die Objekte durch die Partizipation an Umgebungen und die Annahme der Rollen. Sowohl das Betreten als auch Verlassen einer Umgebung kann dynamisch erfolgen. Die Zugehörigkeit eines Objektes zu mehreren Umgebungen ist möglich.
- Die Beschreibung der Trennung von Belangen: Jede Umgebung repräsentiert einen Bereich, sodass die Trennung von Bereichen bei dem Modell explizit unterstützt ist. Die Interaktion zwischen den Bereichen ist durch die gleichzeitige Annahme von Rollen unterschiedlicher Umgebungen realisiert.
- Unterstützung der Wiederverwendbarkeit: Neben den Objekten können auch die Umgebungen mit den darin enthaltenen Rollen als eine Einheit der Wiederverwendung dienen.

Die Sprache “EpsilonJ“

In „EpsilonJ“ nennt man die Umgebungen „contexts“. „Context“ und „role“ sind mit Attributen und Methoden deklariert, so wie Objektklassen. Die Deklaration von „role“ befindet sich innerhalb der Deklaration von „context“, ähnlich den inneren Klassen in der Programmiersprache Java. Instanzen von „contexts“ und „roles“ werden dynamisch erzeugt. Ein Objekt kann dynamisch zu einer Rolle eines Kontextes gebunden und später abgekoppelt werden. Ein Objekt kann außerdem zu mehreren Rollen unterschiedlicher Kontexte gebunden sein. Wenn ein Objekt zu einer Rolle gebunden wird, so kann es die Methoden dieser Rolle ausführen.

Außer den genannten Konzepten wird auch die Möglichkeit der Rollenverwaltung mit XDI untersucht. Welche Technologie dafür tatsächlich im Rahmen dieser Arbeit verwendet wird zeigt sich im Verlauf des Berichts.

3.2 PDS und VRM

Eine der Grundideen dieser Arbeit ist, dass die Institutionen ihre Informationen selbst verwalten. Diese kann zu den Konzepten von PDS und VRM in Verbindung gebracht werden.

3.2.1 VRM

VRM, Vendor Relationship Management, ist ein Konzept, bei dem die Pflege der Beziehungen zu den Unternehmen durch ihre Kunden im Vordergrund steht. VRM gilt als Gegensatz zu CRM. CRM, Customer Relationship Management, ist die Pflege der Beziehungen eines Unternehmens zu seinen Kunden.

Die Entwicklung von VRM basiert auf der Überzeugung, dass freie Kunden wertvoller sind als unfreie. Frei in diesem Sinne beinhaltet folgende Kriterien:

- Beziehungen müssen freiwillig sein;
- Kunden müssen in Beziehungen als unabhängige Akteure agieren;
- Kunden müssen als Punkte der Integration für ihre eigenen Daten sein;
- Kunden müssen die Kontrolle über die Daten haben, die sie generieren und erfassen. Sie müssen in der Lage sein, die Daten selektiv und freiwillig zu teilen und die Nutzung von diesen zu kontrollieren;
- Kunden müssen in der Lage sein, ihre Bedingungen bezüglich der Bindung und des Services geltend zu machen;
- Kunden müssen die Ausdrucksfreiheit bei ihren Forderungen und Absichten ohne Kontrolle irgendeines Unternehmens besitzen.

VRM-Werkzeuge bieten den Kunden zwei Aspekte: Unabhängigkeit von Anbietern und bessere Möglichkeiten beim Kontakt mit den Anbietern. Die gleichen Werkzeuge können außerdem die Individuen bei ihren Beziehungen zu Schulen, Kirchen, Regierungen und anderen Organisationen unterstützen.

Im engeren Sinne ist VRM die Gegenseite, die Kundenseite, von CRM. Im Unterschied zu CRM, bei dem die Pflege der Beziehungen zu den Kunden auf der Seite des Unternehmens liegt, wird bei VRM der Kunde mit den Mitteln ausgestattet, selbst die Beziehungen zum Unternehmen aktiv zu pflegen. Zusammen, VRM und CRM, können diese Werkzeuge für die Verbesserung der Beziehungen zwischen den Kunden und den Unternehmen eingesetzt werden.

VRM-Forschungsgebiet untersucht die Fähigkeit und die Bereitschaft von Kunden, die Unabhängigkeit von Anbietern geltend zu machen und zu nutzen. Auf der anderen Seite untersucht es die Fähigkeit und die Bereitschaft von Anbietern mit den unabhängigen Kunden zu rechnen und in einen Dialog zu kommen.

3.2.2 PDS

Die Abkürzung PDS kann in einem so genannten Personal Data Ecosystem, zu dem dieses Konzept gehört, unterschiedlich interpretiert werden. Die Begriffe werden jedoch nachfolgend abgegrenzt und ihr Zusammenhang erklärt.

- Personal Data Service ist ein Online-Dienst, der es ermöglicht, dass die Individuen ihre Daten unter eigener Kontrolle speichern und verteilen können.
- Personal Data Server ist ein Server, der für die Bereitstellung von Personal Data Services verantwortlich ist.
- Personal Data Store ist eine physische Einheit, eine Datenbank, für die Haltung von persönlichen Daten. Über diese besitzen die Individuen die Zugriffskontrolle.
- Personal Data Ecosystem ist die Gesamtheit von Personal Data Services, Servern und Stores zusammen mit Applikationen, Netzwerken und anderen Komponenten.
- PDX, Personal Data Exchange, ermöglicht die Portabilität von persönlichen Daten zwischen PDS-fähigen Providern und Konsumenten.
- FSW, Federated Social Web, ist die Möglichkeit der Interaktion zwischen Personen und Personen in unterschiedlichen sozialen Netzwerken.

PDS ist ein Dienst zum Speichern von persönlichen Daten. Dabei können die Daten auf einem eigenen Server oder bei einem der PDS Service Providern verwaltet werden. Die Daten befinden sich unter der kompletten Kontrolle von Individuen - Eigentümern von Daten. Dieses Konzept unterstützt die Portabilität von persönlichen Daten zwischen den PDS-fähigen Providern und Konsumenten. Diese standardisierte Austauschfähigkeit nennt man PDX - Personal Data Exchange.

PDS und PDX bilden die benutzerzentrierte Basis für Person-zu-Business-Beziehungen (VRM) und Person-zu-Person-Beziehungen (FSW). Die Überlegungen bei Federated Social Web sind derart, dass die Individuen die PDSs nicht nur für die Interaktion mit den Unternehmen und Organisationen nutzen, sondern auch für die Interaktion untereinander und sogar mit den Benutzern unterschiedlicher sozialer Netzwerke. Dabei bilden sie global verteiltes soziales Netzwerk.

Wie in diesen Konzepten dargestellt, dass den Individuen die Initiative und die Möglichkeit zur Speicherung von und Kontrolle über die eigenen Daten gegeben wird, und außerdem die Unabhängigkeit von anderen Konsumenten, Unternehmen und Organisationen, so ist auch die Idee bei dieser Arbeit den Institutionen die Kontrolle und Verwaltung von ihren Daten zu überlassen.

Die Verwendung von XDI fand bis jetzt u.a. in PDS statt. Es sind aktuell einige Projekte im Gange, die sich mit PDS beschäftigen. Dazu zählen zum Beispiel Projekt Danube oder ein Projekt von Higgins. Das Projekt Nori vereint alle PDS-Projekte, PDS Implementierungen, um Kompatibilität zwischen den Systemen zu unterstützen. Ein Entwickler aus der XRI/XDI Umgebung, namens Markus Sabadello, teilte im Jahr 2010 mit, dass es zurzeit ein großes Interesse an PDS gibt und dieses wächst.

3.3 Dataspaces

Ein anderes verwandtes Thema ist „Dataspaces“. „Dataspaces“ ist ein Konzept für Informationsmanagement [Mic05]. In diesem Abschnitt werden die Grundideen dieses Konzeptes vorgestellt.

Die Entwicklung von relationalen Datenbankmanagementsystemen (DBMSs) war jahrzehntelang der Mittelpunkt im Bereich der Datenverwaltung. In jungen Jahren entwickelte sich aber der Bedarf nach „Daten überall“ bzw. „Daten irgendwo“ und führte so zu den Bemühungen in diesem Problemfeld. Die dringendsten Herausforderungen in der Informationsverwaltung stammen von Organisationen unter Berufung auf eine große Anzahl von diversen, in gegenseitiger Beziehung stehenden, Datenquellen, aber ohne die Möglichkeit ihre „dataspaces“ in praktischer, integrierter Weise zu verwalten. Die „dataspaces“ mit unterstützenden Systemen dienen dabei als eine Möglichkeit der Datenverwaltung. „dataspace“ besteht dabei aus einzelnen Datenquellen.

Das traditionelle DBMS liefert Dienste und Garantien, die es einem Entwickler erlauben, sich auf die Anwendungen zu konzentrieren und nicht auf die Datenverwaltung. Die Situation in der Datenverwaltung sieht heutzutage so aus, dass nur in seltenen Fällen alle Daten in einem DBMS oder einem anderen Single-Datenmodell verwaltet werden können. Stattdessen werden die Entwickler immer häufiger mit einer Reihe von lose-gekoppelten Datenquellen konfrontiert und müssen sich dadurch öfter mit den Herausforderungen der verschiedenen heterogenen Systeme befassen.

Zu diesen Herausforderungen zählen: Unterstützung von Such- und Abfragefähigkeit; Durchsetzung von Regeln, Integritätsbedingungen, Namenskonventionen; Bereitstellung von Verfügbarkeit, Wiederherstellung und Zugriffskontrolle; usw. Solche Herausforderungen sind allgegenwärtig - sie entstehen in unterschiedlichen Organisationen, auf Desktop-PCs oder anderen persönlichen Geräten.

„Dataspaces“ als eine Abstraktion für die Datenverwaltung und die Entwicklung von DataSpace Support Platforms (DSSPs) dienen als Mittelpunkt zur Bewältigung dieser

Herausforderungen. DSSP bietet eine Reihe von miteinander verbundenen Diensten und Garantien, die es ermöglichen, dass sich Entwickler auf die spezifischen Herausforderungen ihrer Anwendungen konzentrieren, anstatt auf wiederkehrende Herausforderungen im Umgang mit großen Datenmengen, die miteinander in Beziehung stehen, aber ungleichartig verwaltet werden.

Eigenschaften von Dataspace-Systemen sind:

- DSSP arbeitet mit Daten und Applikationen in einer Vielzahl von Formaten über mehrere Systeme mit unterschiedlichen Schnittstellen. DSSP ist erforderlich, um alle Daten in einem „dataspace“ zu unterstützen.
- Im Unterschied zu einem DBMS hat DSSP nicht die volle Kontrolle über die Daten.
- Abfragen zu DSSP können unterschiedliche Ebenen von Diensten liefern, und in einigen Fällen ungefähre Antworten geben.
- DSSP muss Werkzeuge anbieten, um eine enge Integration von Daten in „space“, so wie es notwendig ist, zu schaffen.

„Dataspaces“ verfolgen nicht den Daten-Integrations-Ansatz, sondern mehr Daten-Ko-Existenz-Ansatz. Das Ziel dabei ist die Bereitstellung von Basisfunktionalitäten über alle Datenquellen ohne Rücksicht auf den Integrationsgrad von diesen.

Dieses Konzept setzt keine volle Kontrolle über die Daten in „dataspace“ voraus, sondern erlaubt den teilnehmenden Systemen die Daten zu verwalten und unterstützt neue Dienste, die bei der Aggregation von Systemen entstehen.

3.4 Zusammenfassung

In diesem Kapitel wurden die verwandten Arbeiten vorgestellt. Das sind Rollenmodelle, Dataspaces, PDS und VRM. Bei den Rollenmodellen geht es darum, dass die Aktoren in unterschiedlichen Kontexten in unterschiedlichen Rollen auftreten können. Der Abschnitt über Rollenmodelle beschreibt die Ansätze für die Rollenverwaltung und die technische Realisierung von diesen. Dataspaces ist ein Konzept für die Arbeit mit unterschiedlichen Datenquellen. Dabei werden diese Datenquellen unterschiedlich verwaltet. Dataspaces bietet eine Abstraktionsschicht über diesen Datenverwaltungen. Dabei wird kein Integrationsansatz sondern der Ko-Existenz-Ansatz verfolgt. Die Entwickler können sich bei diesem Ansatz um ihre Anwendungen und müssen sich nicht um die unterschiedlichen

Eigenschaften von den Datenquellen kümmern. VRM liefert eine Idee, bei der die Kunden die Beziehungen zu den Unternehmen selbst aktiv pflegen. PDS ist ein Dienst zum Speichern von persönlichen Daten. Dabei können die Daten auf einem eigenen Server oder bei einem der PDS Service Providern verwaltet werden. Die Daten befinden sich unter der kompletten Kontrolle von Individuen - Eigentümern von Daten.

Im Rahmen dieser Arbeit müssen die Rollen von Aktoren realisiert werden. Dazu werden die Rollenkonzepte dieses Abschnitts und außerdem die Verwaltung von Rollen durch das im Kapitel über Grundlagen vorgestellte XDI-Konzept untersucht. Wie in VRM und PDS dargestellt, dass den Individuen die Initiative und die Möglichkeit zur Speicherung von und Kontrolle über die eigenen Daten gegeben wird, und außerdem die Unabhängigkeit von anderen Konsumenten, Unternehmen und Organisationen, so ist auch die Idee bei dieser Arbeit den Institutionen die Kontrolle und Verwaltung von ihren Daten zu überlassen.

4 Lösungsansatz

Bevor mit der Lösungsbeschreibung begonnen wird, soll hier in den folgenden Kapiteln der Lösungsansatz kurz skizziert werden.

4.1 Fachkonzept

Das Resultat dieses Projektes, alpha-Institutions, das als Teilprojekt von alpha-Flow [Chr09] gilt, ist die Implementierung einer Software. Diese Software soll ermöglichen, dass die Institutionen die Daten, die die Institution selbst beschreiben, autonom verwalten und untereinander austauschen. Die Idee dabei ist, dass die Gesamtheit aller Informationen über Institutionen, Aktoren oder Ähnliches so auf alle Institutionen verteilt werden, dass jede Institution für eigene Informationen selbst zuständig ist und diese bei Bedarf bereitstellt. alpha-Flow beschäftigt sich mit aktiven Dokumenten, die in unterschiedlichen Szenarien zwischen den Institutionen bzw. den Aktoren in diesen Institutionen ausgetauscht werden. Dabei erfragen die Aktoren bestimmte Informationen von möglichen kooperierenden Institutionen, wie z.B. bestimmte in den Institutionen arbeitende Spezialisten, um dann ein aktives Dokument an diese zu senden. Die Aufgabe von alpha-Institutions an dieser Stelle ist, den Aktoren bei Bedarf die Informationen über kooperierende Institutionen bereitzustellen.

Die zu entwerfende und implementierende Software muss eine Reihe an Anforderungen erfüllen. Sie muss administrative Funktionalitäten bereitstellen. Darunter fallen die Speicherung, die Modifikation sowie das Entfernen von Daten. Das sind die Daten, die die Institutionen beschreiben. Da die Institutionen für die Verwaltung ihrer Daten selbst verantwortlich sind, sind diese Daten verteilt. Jede Institution verwaltet ihre Daten auf dem eigenen Server. Die Software muss also die Verteilung der Daten berücksichtigen. Im Fokus stehen solche Daten, die die Aktoren mit ihren Rollen, die eine Spezialisierung des Aktors bedeuten, in den jeweiligen Institutionen beschreiben. Diese Daten werden durch digitale Identifikatoren identifiziert. Falls eine Institution die Pflege des eigenen Servers nicht wünscht, kann diese ihre Daten auf dem übergeordneten Server namens alphaServer pflegen. Die Verwendung vom alphaServer wird im folgenden Kapitel erklärt.

Außerdem muss die Software in der Lage sein, die Suchanfragen auszuführen. Von Bedeutung sind solche Anfragen, die z.B. alle Aktoren einer Institution, alle Institutionen, alle Aktoren zu einer bestimmten Rolle oder Ähnliches als Suchkriterium haben. Dabei ist zu beachten, dass in einer Institution die Aktoren in unterschiedlichen Rollen auftreten können.

Das Bild 4.1 zeigt ein Anwendungsfalldiagramm, in dem die Hauptfunktionalitäten des Systems grob dargestellt werden.

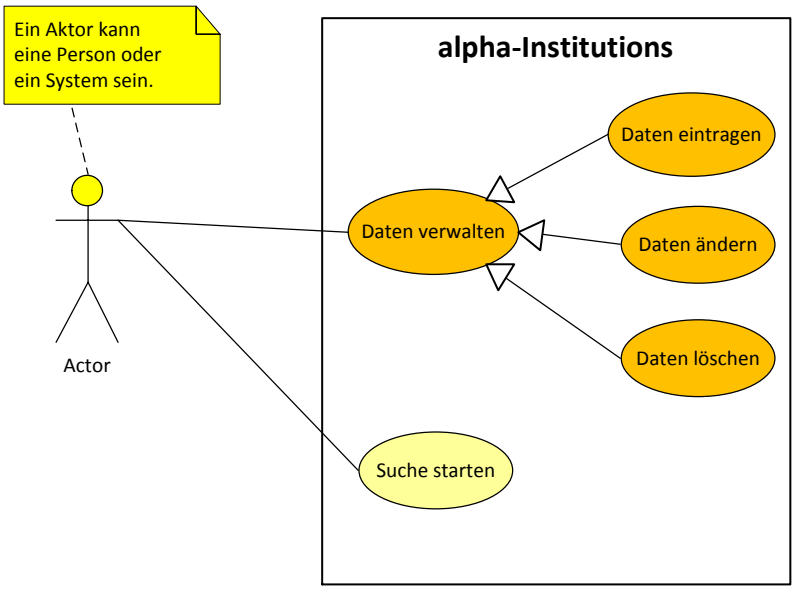


Bild 4.1: Use Cases

Da es immer die Möglichkeit besteht, dass neue Institutionen hinzukommen, muss bei der Umsetzung die Skalierbarkeit berücksichtigt werden. Um die Autonomie der Institutionen zu ermöglichen, muss die Software außerdem verteilt realisiert werden.

Um die Aufgabe zu lösen muss man sich zuerst Gedanken darüber machen, was bei einer verteilten Institutionsverwaltung zu beachten ist. Als Erstes wird überlegt, wie man die Institutionen und auch Aktoren identifiziert. Desweiteren ist zu bedenken, wie der Datenaustausch bzw. die Datenverwaltung erfolgt.

4.2 Grobarchitektur

Die genannten Probleme werden mithilfe sehr junger Technologien, XRI und XDI, auf die in der Aufgabenstellung als vielversprechend hingewiesen wurde, gelöst. XRI dient in

dieser Arbeit der Identifikation und Lokalisierung von Institutions- bzw. Aktoreninstanzen. Mit XDI werden die Verwaltung und der Austausch von Daten realisiert.

Das gesamte System ist in einem Client-Server-Modell organisiert. Im verteilten Szenario verwaltet jede Institution ihre Daten auf dem eigenen Server. Jede Institution hat dabei eigene Client und Server. Das Bild 4.2 zeigt den Aufbau des Systems in dieser Variante und gleichzeitig ein Beispiel für die Kommunikation von Komponenten.

Das Bild 4.3 zeigt den alternativen Aufbau des Systems. In dieser Variante pflegt jede Institution ihre Daten nicht auf dem eigenen, sondern auf einem übergeordneten Server. In diesem Fall auf dem alphaServer. Der alphaServer spielt in diesem System eine bestimmte Rolle. Der zentrale alphaServer ist notwendig, um die Liste aller beteiligten Institutionen und weiterer möglicher Kataloge zu verwalten.

Der Client besteht aus zwei Komponenten, AlphaQClient für die Suchanfragen und AlphaDataAdministrationClient für das Einfügen, Ändern und Löschen von Daten. Ein Client ist in der Lage, mit dem Server jeder Institution zu kommunizieren und ihm Nachrichten zu senden. Im folgenden Verlauf werden die eben dargestellten Konzepte anhand von Bildern 4.2, 4.3 und 4.4 näher erläutert.

Zunächst wird die Bedeutung vom alphaServer erklärt. Im verteilten Szenario, bei dem alle Institutionen ihre Daten auf einem eigenen Server verwalten, muss die Information über alle teilnehmenden Institutionen irgendwo verwaltet werden. Eine Idee wäre, diese Information bei allen Institutionen zu speichern. Dies würde aber einen gewissen Synchronisationsaufwand an sich ziehen. Stattdessen wird diese Information ausgelagert und auf dem übergeordneten Server, alphaServer, geführt. Jederzeit kann eine beliebige Institution diese Information abfragen. Die Liste der Institutionen ist unter „institutions“ abgelegt. „institutions“ ist ein Subsegment von der XRI „@alpha*institutions“. Außerdem können auf dem alphaServer weitere Informationen abgelegt werden. So werden die möglichen Rollen von Aktoren in den Institutionen unter „rolecatalog“ geführt. „rolecatalog“ ist ein Subsegment von der XRI „@alpha*stdcatalogs*rolecatalog“. Die Einträge unter „institutions“ und „rolecatalog“ werden von den beiden Client-Komponenten verwendet. AlphaQClient verwendet diese bei Suchanfragen. AlphaDataAdministrationClient ändert oder löscht bei Bedarf die Einträge in diesen Listen oder fügt neue Einträge hinzu. So kann zum Beispiel eine Institution die Liste mit möglichen Rollen beim alphaServer anfragen. Dies zeigt das Bild 4.4.

Folgend wird anhand vom Bild 4.2 ein möglicher Aufbau des Systems, bei dem jede Institution die Daten auf einem eigenen Server pflegt, erklärt. Jeder Server, einschließlich alphaServer, hat drei Komponenten: alpha (www.alpha.com), resolve (resolve.alpha.com)

und xdi (xdi.alpha.com). alpha-Komponente ist für die Registrierung und Verwaltung von XRIs zuständig. Wie man XRIs registriert und verwaltet liefert der Anhang zu dieser Arbeit. resolve ist für die Auflösung von XRIs zuständig. Kapitel 2.4 liefert nähere Informationen, was die Auflösung bedeutet und wie XRIs aufgelöst werden. xdi ist die Komponente, die das Arbeiten mit den Daten ermöglicht. Es wird hier angenommen, dass XRIs bereits registriert sind. Außerdem wird die Auflösung nicht detailliert erläutert, sondern für das bessere Verständnis der Aufbau und die Kommunikation grob erklärt. Die Kommunikation erfolgt über das Internet. Alle Daten sind in XDI gespeichert. Falls man mit den Daten arbeiten will, sei es die Suche oder die Modifikation, muss der Client mit der XDI-Komponente kommunizieren. Der Client muss dabei wissen, wo sich diese Komponente im Netzwerk befindet. Dafür muss der Client jedes Mal, bevor er mit der XDI-Komponente irgendeines Servers kommuniziert, die entsprechende XRI auflösen. Für die Auflösung ist die resolve-Komponente zuständig. Diese Komponente liefert zu einer XRI die Netzwerk-Adresse von der xdi-Komponente. Im folgenden Beispiel wird gezeigt, wie die Kommunikation stattfindet, falls man eine Liste mit allen Radiologen aller Institutionen erhalten will (Bild 4.2).

1. Im ersten Schritt (nachdem die entsprechende Adresse durch die Auflösung ermittelt wurde) sendet der Client (AlphaQClient-Komponente) zu der xdi-Komponente vom alphaServer die Anfrage für die Liste der beteiligten Institutionen.
2. Nachdem diese als Antwort erhalten wurde, sendet der Client iterativ jeweils eine Anfrage an die xdi-Komponente der Server aller Institutionen für die Liste aller Radiologen in den jeweiligen Institutionen. Hier ist zu bemerken, dass der Client vor jeder Anfrage an eine Institution die Adresse der xdi-Komponente ermittelt.
3. Nachdem der Client die Antwort von allen Institutionen erhalten hat, fügt er alle Listen mit Radiologen zusammen und gibt die erstellte Liste mit allen Radiologen zurück.

Falls die Institutionen keine Server pflegen wollen, können sie ihre Daten auch auf dem alphaServer verwalten. Das Bild 4.3 zeigt diese Möglichkeit. Die Vorgehensweise beim Datenaustausch ist wie in der Variante, bei der jede Institution eigenen Server pflegt. Alle Anfragen werden allerdings nur zu dem alphaServer gesendet. Die beiden Varianten funktionieren gleichermaßen, da es für den Client keine Rolle spielt, auf welchem Server sich die Daten befinden. Er bekommt nach der Auflösung die Adresse von der xdi-Komponente und sendet die Anfragen an diese.

Es ist auch möglich, dass ein Teil der Institutionen eigenen Server pflegt und ein anderer Teil eigene Daten auf dem alphaServer verwaltet. Der Client ist so realisiert, dass auch diese Variante funktionieren würde.

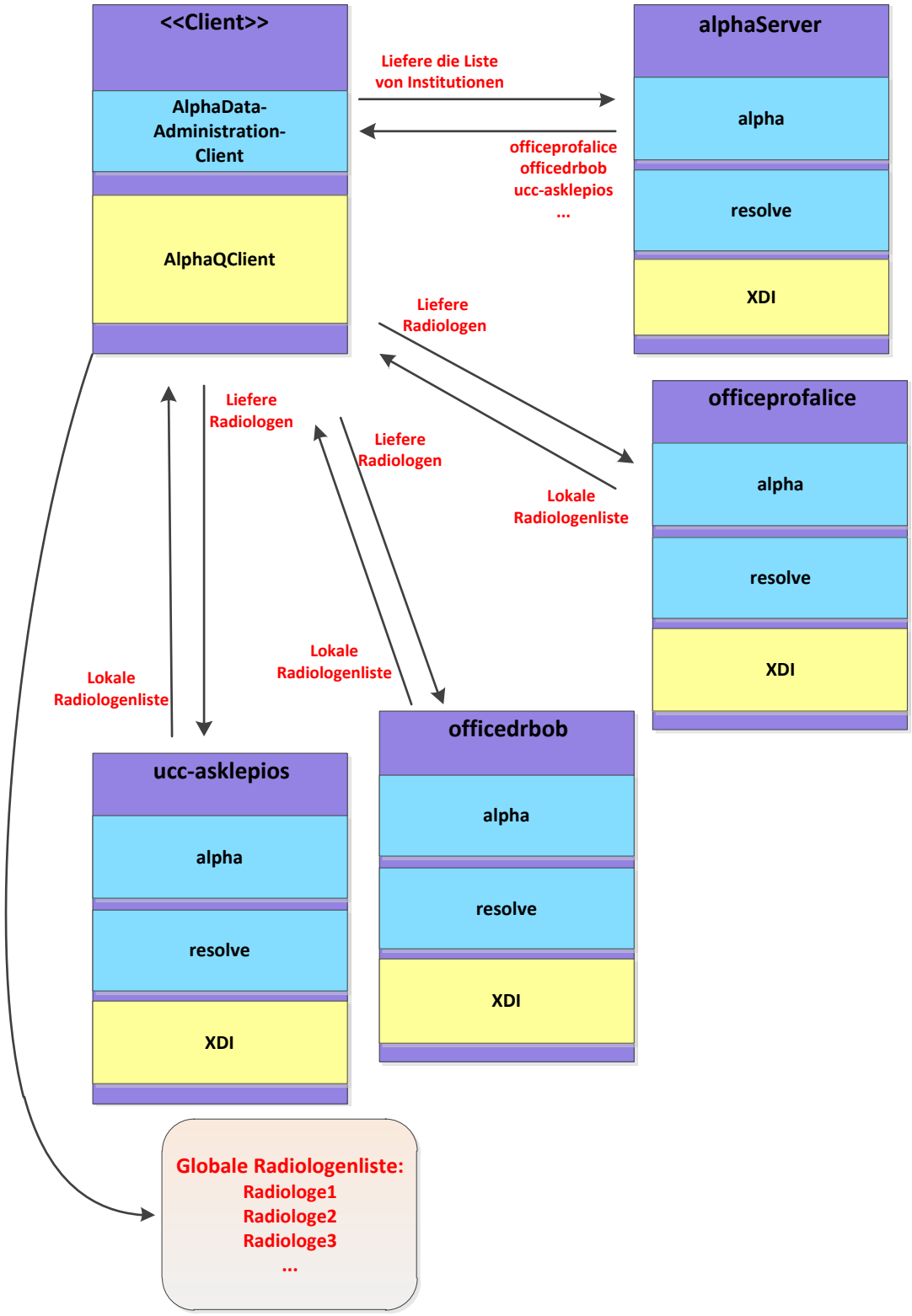


Bild 4.2: Verteiltes Scenario

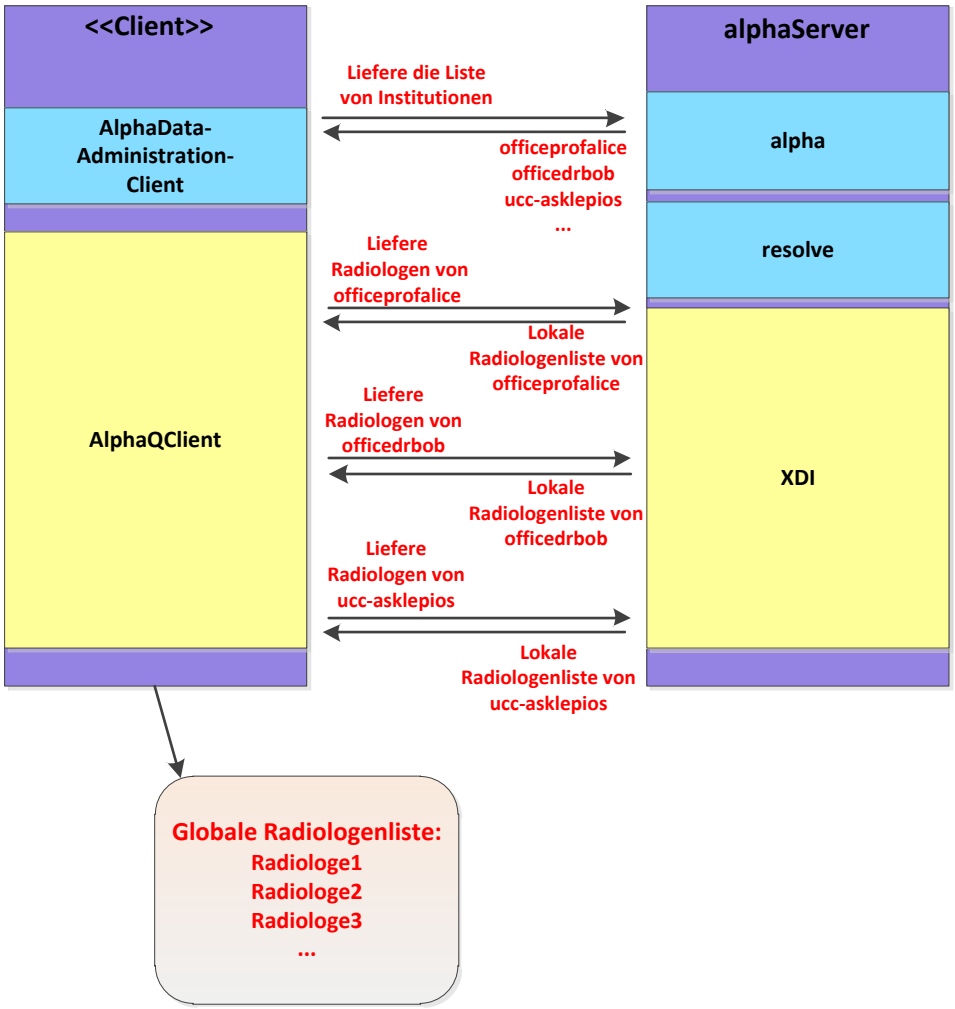


Bild 4.3: Alternative mithilfe von alphaServer

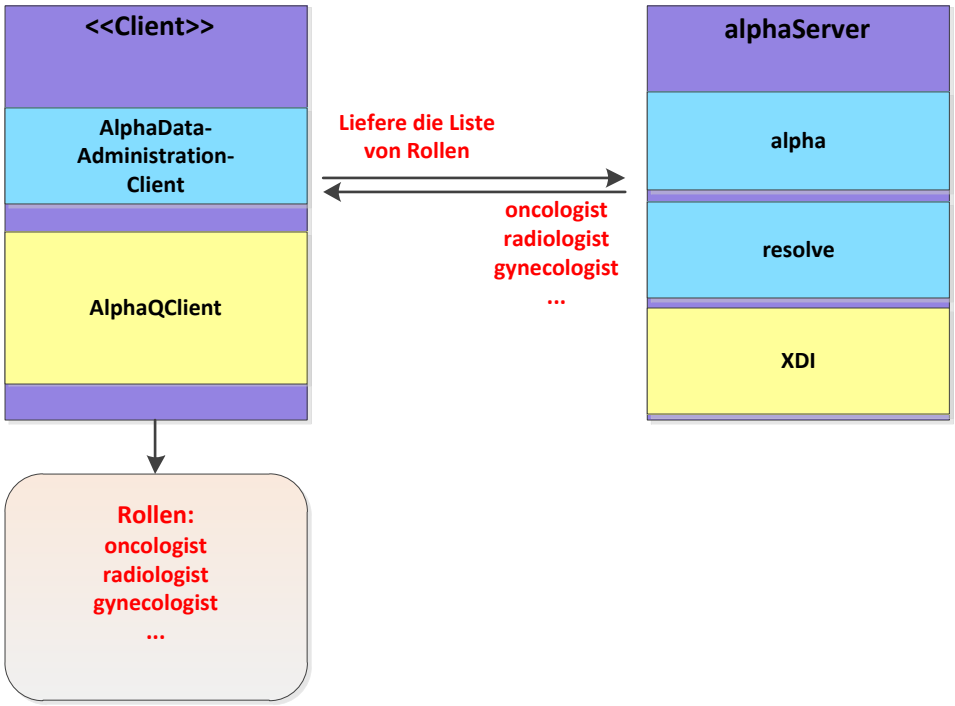


Bild 4.4: Rollen

4.3 Zusammenfassung

Die zu entwickelnde Software muss also den Institutionen die Möglichkeit geben ihre Institutionsdaten selbständig und autonom zu verwalten. Außerdem muss sie ermöglichen, dass die Institutionen diese Informationen untereinander austauschen können. Dafür hat jede Institution einen Server für die Verwaltung von Daten und einen Client für den Datenaustausch. Der alphaServer dient als zentrale Registratur von Institutionen, damit diese voneinander erfahren können.

5 Systementwurf

Die Grundidee der Aufgabe ist es, den beteiligten Institutionen die Verwaltung ihrer Daten zu überlassen. Dies sollte autonom und verteilt geschehen.

5.1 Server

Um bereits beschriebenes Ziel zu erreichen, betreibt jede Institution ihren eigenen Server, auf dem die Daten abgelegt und verwaltet werden. Damit die Institutionen jederzeit von allen aktuell beteiligten Institutionen erfahren können, muss ein Katalog geführt werden. Dazu dient ein alphaServer, der für diesen und auch weitere mögliche Kataloge zuständig ist.

Auf allen Servern läuft die Software namens ibrokerKit. Diese Software wurde von den Entwicklern aus der XRI/XDI-Umgebung entwickelt und für die Bearbeitung der Aufgabe freundlicherweise zur Verfügung gestellt. ibrokerKit benötigt selbst OpenXRI, einen Datenbank-Server und einen Servlet-Container. Dabei wurden im Rahmen dieser Arbeit MySQL und Apache Tomcat verwendet. In MySQL werden die XRIs und die Benutzerdaten abgelegt. Mithilfe von OpenXRI verwaltet ibrokerKit die XRIs. Wie man MySQL und Tomcat einrichtet, und danach ibrokerKit zusammen mit OpenXRI installiert und startet liefert die Anleitung zu ibrokerKit. Die entsprechenden Informationen liefert außerdem der Anhang dieses Berichtes.

Es wurden folgende Versionen/Revisionen verwendet:

- openxri: Version 1.2.2/Revision 484
- ibrokerkit: Version 0.5/Revision 589
- MySQL 5.1
- Tomcat 6.0.29

Für die Bearbeitung der Aufgabe wurden einige Teile von ibrokerKit umbenannt. Die Namen mit der Bezeichnung „youribroker“ in verschiedenen Komponenten wurden durch „alpha“ ersetzt. Zum Beispiel xdi.youribroker.com durch xdi.alpha.com.

Unter Tomcat laufen die Komponenten resolve.alpha.com, www.alpha.com und xdi.alpha.com. resolve.alpha.com ist für die Auflösung von XRIs, www.alpha.com sowohl für die Registrierung XRIs als auch für die Verwaltung von XRIs und Benutzerdaten, xdi.alpha.com für die Arbeit mit den XDI-Daten zuständig.

Jede XRI, die über ibrokerKit angelegt wird, besitzt einen sogenannten XDI-Endpoint, der unter einer bestimmten Adresse zu erreichen ist. Die Clients kommunizieren mit solchen Endpoints, wenn sie mit den XDI-Daten arbeiten wollen. Die XDI-Daten im Kontext dieser Arbeit sind nichts anderes als die Institutions- bzw. Aktoreninformationen, die in einem XDI-Dokument gespeichert sind. Das heißt - zu jeder XRI existiert ein XDI-Endpoint, mit dem man kommuniziert, um mit den XDI-Daten der betreffenden XRI zu arbeiten.

Das Bild 5.1 zeigt die Komponenten des Systems, die auf dem Server laufen. Da der Aufbau bei allen Servern, alphaServer oder Institutionsserver, gleich ist, wird der Aufbau an einem Server gezeigt.

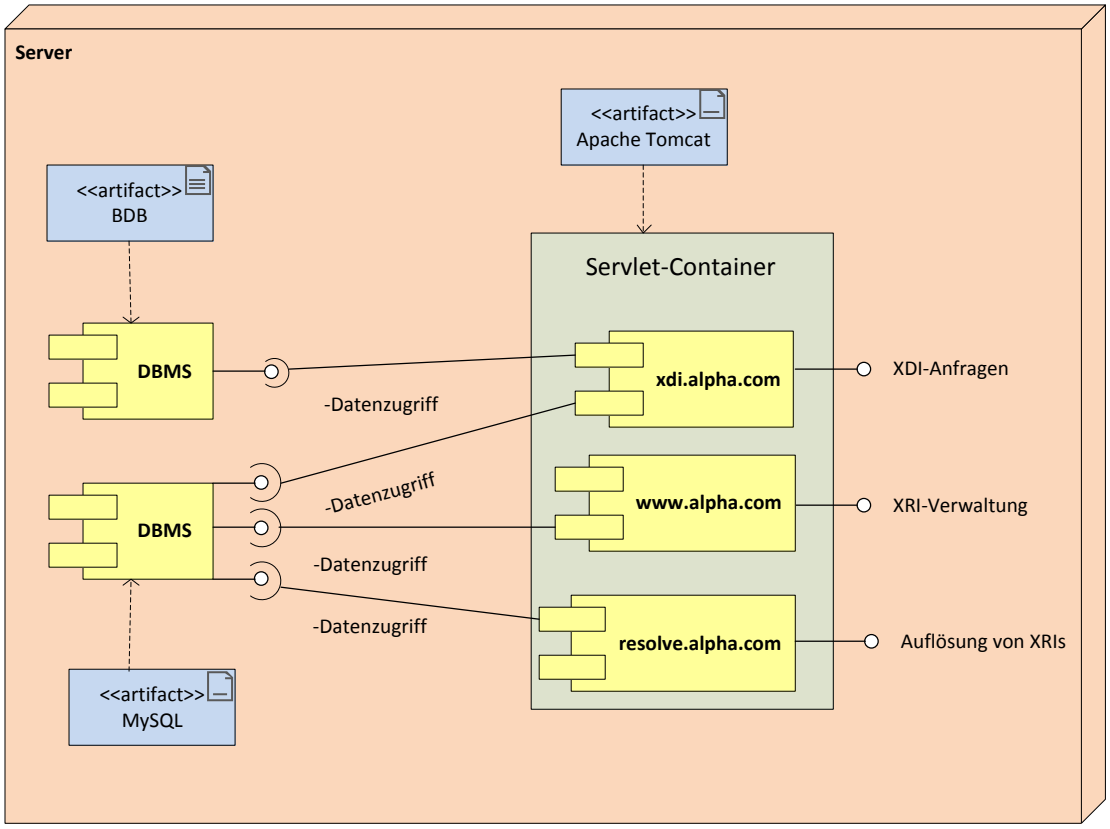


Bild 5.1: Serverkomponenten

5.2 Client

Jede Institution und auch Alpha-Administration erhält einen Client, der Funktionen für die Arbeit mit den XDI-Daten enthält. Der Client ist in vier Klassen unterteilt. AlphaResolver dient der Auflösung. Er liefert zu einer gegebenen XRI die Adresse von dem XDI-Endpoint, der zu dieser XRI gehört. AlphaMessenger kommuniziert mit einem XDI-Endpoint und tauscht mit ihm die Nachrichten aus. Dabei verwendet diese Klasse den AlphaResolver, um den XDI-Enpoint zu bestimmen. AlphaDataAdministrationClient ist für die Verwaltung von Institutionsinformationen zuständig. Diese Klasse liefert die Funktionen zum Erstellen, Modifizieren und Löschen von Daten. AlphaQClient liefert die Funktionen für verschiedene Suchoperationen. Beide letztgenannten Klassen verwenden die Klasse AlphaMessenger, um Nachrichten an die XDI-Endpoints zu senden.

5.2.1 CRC

Nachfolgend werden diese Klassen mit ihren oben beschriebenen Verantwortlichkeiten in Form von CRC Cards, Bild 5.2, gezeigt.

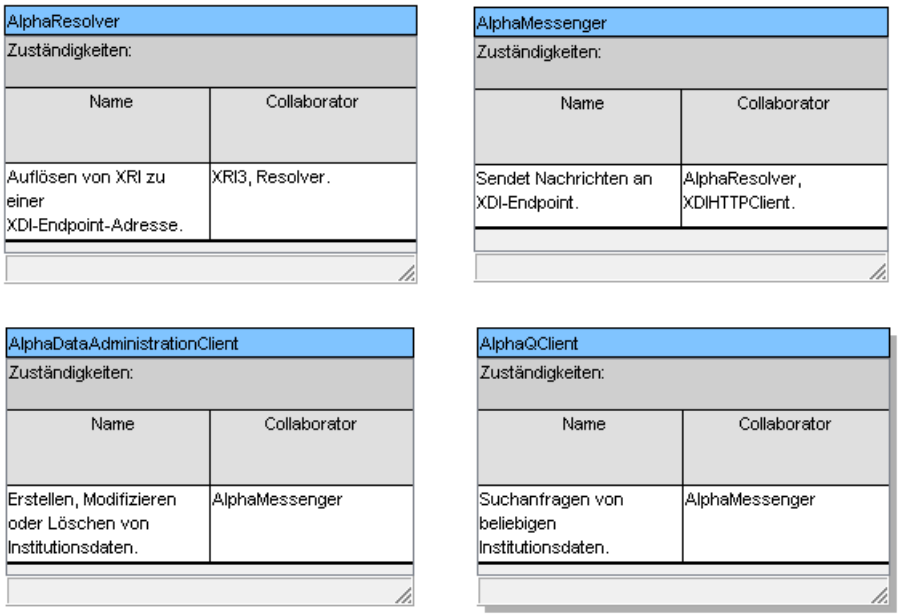


Bild 5.2: CRC Cards Diagram

5.2.2 Sequenzdiagramme

Die Sequenzdiagramme, Bild 5.3 und Bild 5.4, zeigen außerdem die Interaktionen zwischen den Klassen.

Im Bild 5.3 wird dargestellt, wie eine Suchanfrage stattfindet:

1. Ein Aktor startet die Suchanfrage, indem er ein Objekt der Klasse AlphaQClient erzeugt und anschließend eine der Suchfunktionen wie zum Beispiel getAllRoles() aufruft.
2. Danach erzeugt AlphaQClient ein Objekt der Klasse AlphaMessenger und ruft deren Funktion messageToXDIEndpoint() auf, um eine Suchanfrage an den XDI-Endpoint zu senden.
3. Bevor der AlphaMessenger die Anfrage sendet, ermittelt er zunächst die URI von dem entsprechenden XDI-Endpoint. Dafür verwendet er die Klasse AlphaResolver, indem er ein Objekt dieser Klasse erzeugt und deren Funktion findXDIURI() aufruft.
4. Anschließend sendet der AlphaMessenger die Suchanfrage mithilfe von der Klasse XDIClient an die ermittelte XDI-URI.

Das Bild 5.4 zeigt nahezu gleichen Ablauf mit dem einzigen Unterschied, dass anstatt von der Klasse AlphaQClient mit ihren Suchfunktionen die Klasse AlphaDataAdministrationClient mit Funktionalitäten zur Administration von Daten verwendet wird.

5.2.3 Schnittstellenklassen

Die Client-Klassen liefern die Funktionen zum Einfügen, Löschen, Modifizieren oder Suchen von Daten. Dabei erwarten diese Funktionen je nach Aufgabe unterschiedliche Parameter. Alle Parameter verschiedener Funktionen haben den Typ „String“. Mögliche Parameter sind entweder XRIs oder XDI-Elemente bzw. Elemente des XDI-Graphen. Kapitel 2.3 erläutert das Konzept des XDI-Graphen mit seinen Elementen. Die beiden Schnittstellenklassen des Clients werden als Klassendiagramme anhand von den Bildern 5.5 und 5.6 dargestellt.

Folgend werden die Methoden von AlphaQClient und anschließend von AlphaDataAdministrationClient erklärt. AlphaQClient liefert die Funktionen zum Suchen und AlphaDataAdministrationClient zum Verwalten von Daten. Alle Methoden von AlphaQClient haben als Rückgabe ein Array vom Typ „String“.

AlphaQClient:

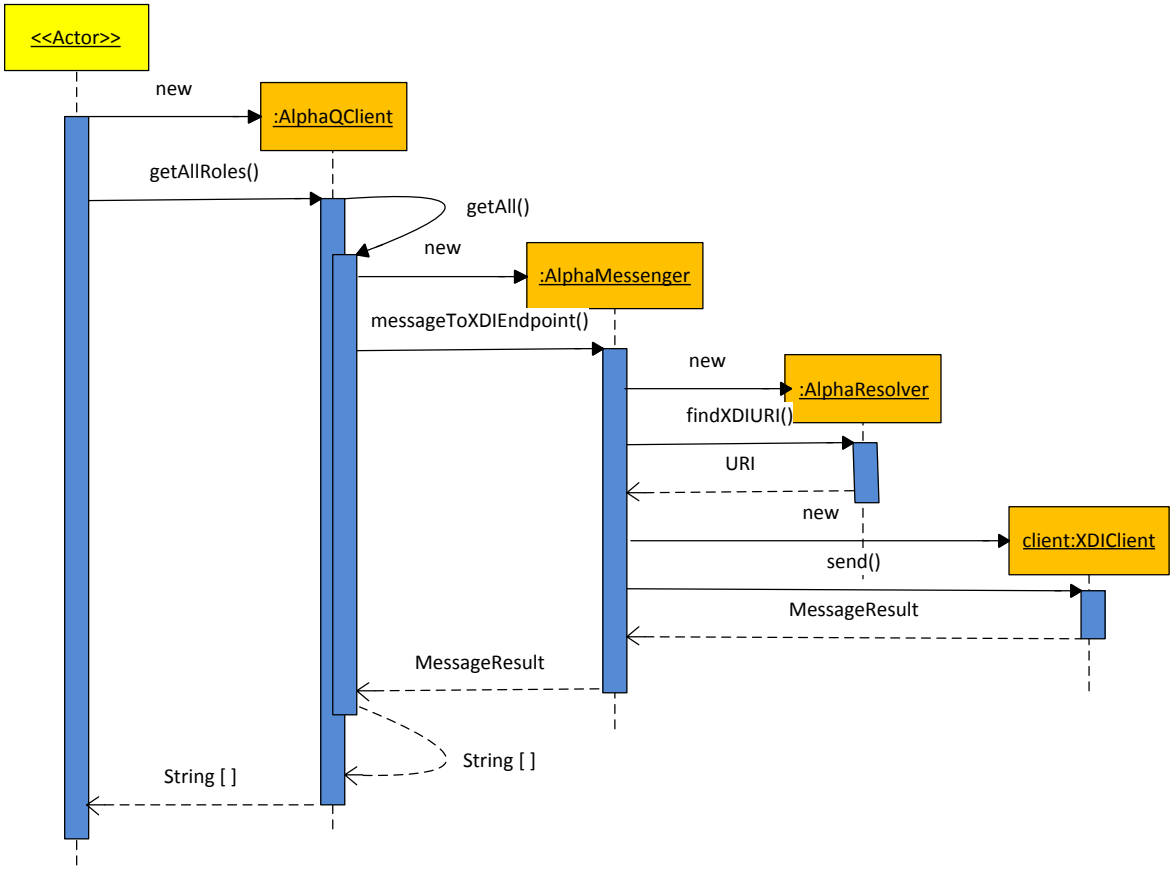


Bild 5.3: AlphaQClient

- String[] getAllInstitutions() - gibt alle Institutionen zurück.
- String[] getAllActorsFromInstitution(String iname) - gibt zu einer Institutions-XRI (iname) alle Aktoren von dieser Institution zurück.
- String[] getAllRolesFromActor(String iname) - gibt zu einer Aktor-XRI (iname) alle Rollen von diesem Aktor zurück.
- String[] getAllActorsToRole(String iname) - gibt zu einer Rolle (iname) alle Aktoren, die diese Rolle besitzen, zurück.
- String[] getAllRoles() - gibt alle möglichen Rollen zurück.
- String[] getAll(String iname) - universelle Methode. Gibt zu einer beliebigen XDI-Adresse die Einträge, die darunter gespeichert sind, zurück.

AlphaDataAdministrationClient:

- In allen Funktionen gilt - Parameter für XRI ist gleich das Subjekt im XDI-Graphen. Ein Objekt ist entweder ein Literal oder eine Referenz.

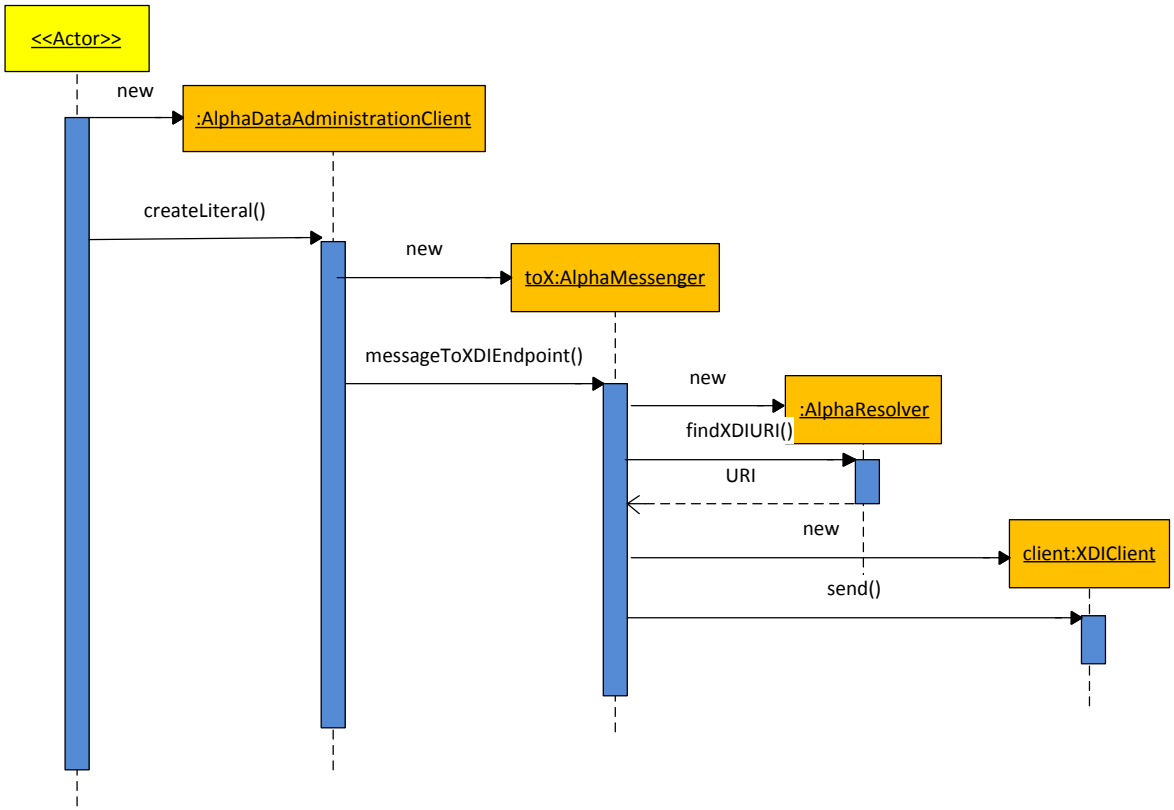


Bild 5.4: AlphaDataAdministrationClient

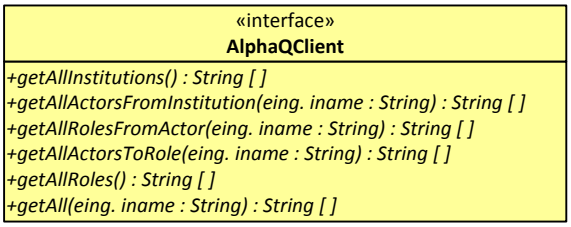


Bild 5.5: AlphaQClient

- void createSubject(String subject, String xri3) - Erstellt ein Subjekt im XDI-Graphen zu der XRI xri3.
- void createPredicate(String predicate, String xri3) - Erstellt ein Prädikat unter dem Subjekt. Falls das Subjekt nicht existiert, wird dieses ebenfalls erstellt. Parameter xri3 ist eine gültige XRI und gleichzeitig ein Subjekt.
- void createReference(String reference, String predicate, String xri3) - Erstellt ein Objekt unter dem Subjekt/Prädikat. Falls das Subjekt und das Prädikat nicht existieren, werden diese ebenfalls erstellt.

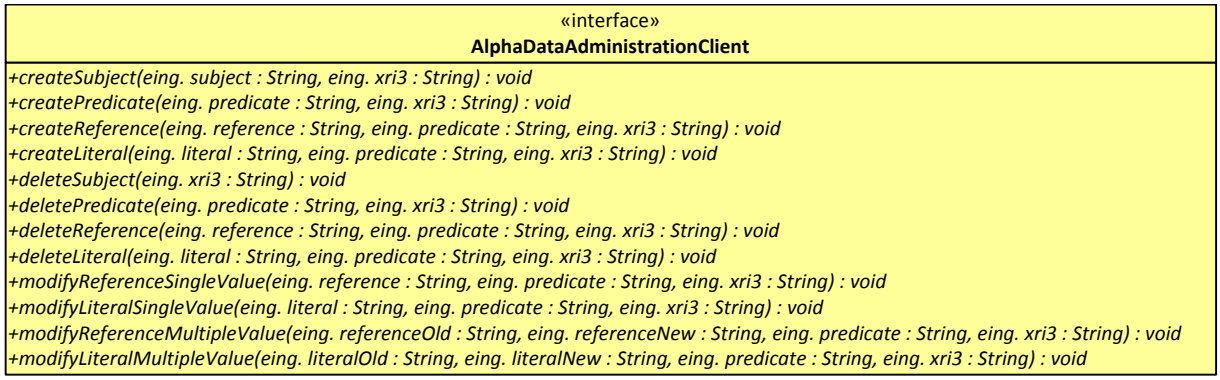


Bild 5.6: AlphaDataAdministrationClient

- void createLiteral(String literal, String predicate, String xri3) - Erstellt ein Objekt unter dem Subjekt/Prädikat. Falls das Subjekt und das Prädikat nicht existieren, werden diese ebenfalls erstellt.
- void deleteSubject(String xri3) - Entfernt ein Subjekt. Alles, was unter dem Subjekt gespeichert ist, wird ebenfalls entfernt.
- void deletePredicate(String predicate, String xri3) - Entfernt ein Prädikat unter dem Subjekt. Alles, was unter dem Prädikat gespeichert ist, wird ebenfalls entfernt.
- void deleteReference(String reference, String predicate, String xri3) - Entfernt ein Objekt unter dem Subjekt/Prädikat.
- void deleteLiteral(String literal, String predicate, String xri3) - Entfernt ein Objekt unter dem Subjekt/Prädikat.
- void modifyReferenceSingleValue(String reference, String predicate, String xri3) - Ändert ein Objekt. Gilt nur für Einträge, bei denen unter einem bestimmten Prädikat nur ein Objektwert gespeichert ist. Z.B. eine Adresse.
- void modifyLiteralSingleValue(String literal, String predicate, String xri3) - Ändert ein Objekt. Gilt nur für Einträge, bei denen unter einem bestimmten Prädikat nur ein Objektwert gespeichert ist. Z.B. "Name".
- void modifyReferenceMultipleValue(String referenceOld, String referenceNew, String predicate, String xri3) - Ändert ein Objekt. Gilt für beliebige Einträge.
- void modifyLiteralMultipleValue(String literalOld, String literalNew, String predicate, String xri3) - Ändert ein Objekt. Gilt für beliebige Einträge.

Die Implementierungen von den Schnittstellenklassen sind AlphaQClientImpl und AlphaDataAdministrationClientImpl.

5.3 Client+Server

Bei der Abwicklung von Aufgaben - die Suche mit dem AlphaQClient, die Datenverwaltung mit dem AlphaDataAdministartionClient und die XRI-Verwaltung mit dem Webbrowser - kommuniziert der Client mit dem Server, der ein Server der XRI-Infrastruktur, der alphaServer oder ein Institutionsserver sein kann. Das Bild 5.7 veranschaulicht grob das Zusammenspiel von Client und Server bei dieser Kommunikation. Dabei lösen sowohl AlphaQClient als auch AlphaDataAdministrationClient zuerst die XRIs über die resolve-Komponente auf. Anschließend kommunizieren sie mit der xdi-Komponente, um Nachrichten an diese zu senden. Die Verwaltung von XRIs geschieht mithilfe von einem Webbrowser. Dabei kommuniziert dieser mit der alpha-Komponente.

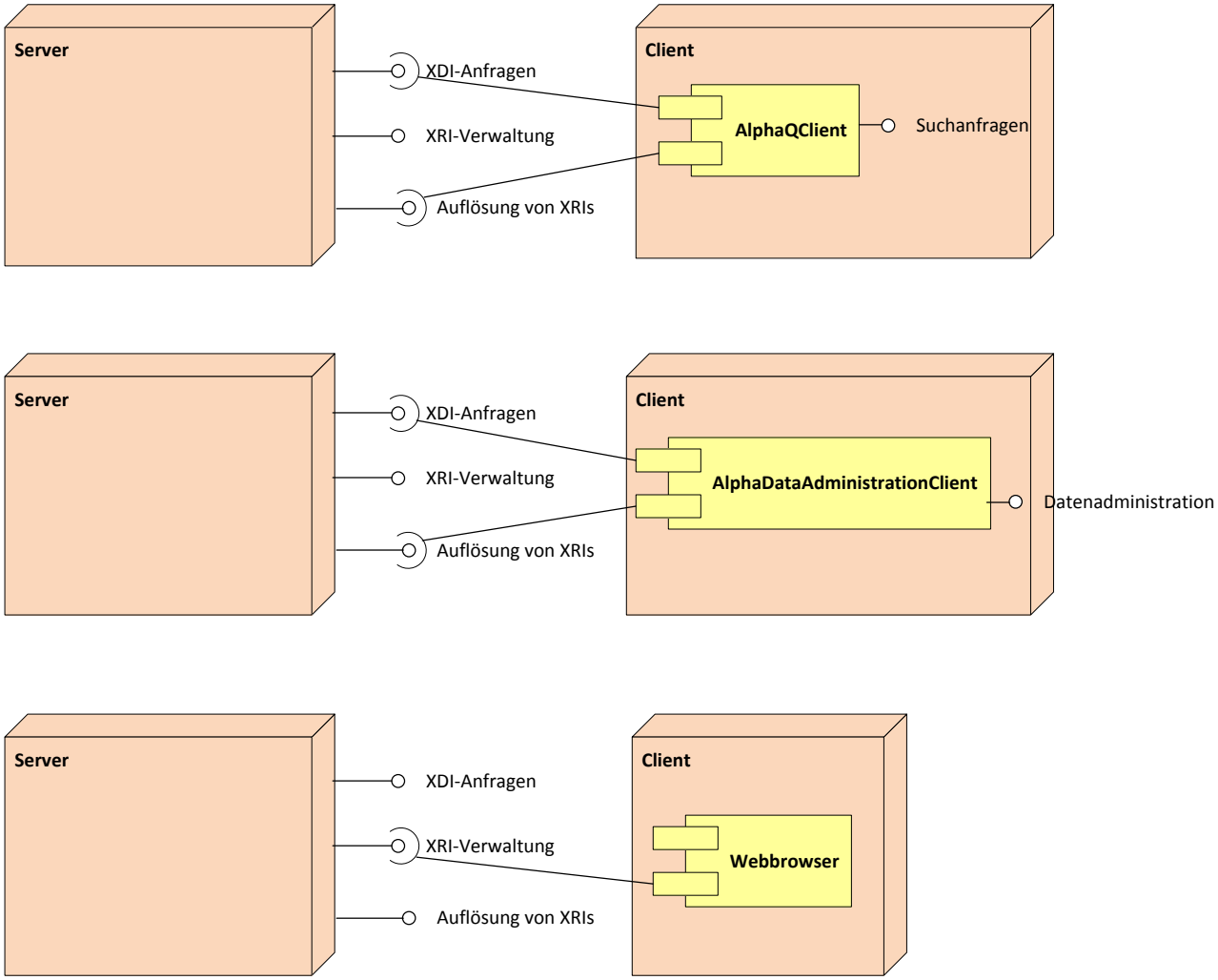


Bild 5.7: Zusammenspiel von Client und Server

5.4 Zusammenfassung

Die Serverseite ist mittels eines Softwaresystems namens ibrokerKit aufgebaut. Von den zahlreichen Komponenten von ibrokerKit werden hier solche verwendet, die für die Auflösung von XRIs, die Verwaltung von XRIs und das Arbeiten mit den XDI-Daten, die zu einer bestimmten XRI gehören, dienen. Die Clientseite ermöglicht die administrative Arbeit mit den Institutionsdaten und außerdem stellt Funktionen bereit, die beliebige Suchanfragen bereitstellen.

6 Technische Umsetzung

Nachdem der Lösungsansatz und der Entwurf erklärt wurden, werden hier nun einige Teile der Realisierung dargestellt. Auf der Serverseite wird gezeigt, wie die XRI-Infrastruktur aufgebaut ist. Auf der Clientseite, wie die Funktionalitäten zum Arbeiten mit XDI-Daten, Institutionsinformationen, implementiert sind.

6.1 Server

Zuerst wurde global eine XRI, ein i-name, @alpha angelegt. Bei dieser XRI wurde außerdem ein Eintrag gemacht, der die URI enthält, an der der alphaServer läuft. Genauer gesagt, die Komponente von alphaServer, die für die Auflösung von Subsegmenten von @alpha verantwortlich ist. alphaServer dient als zentrale Registratur von Institutionen und ist der Community-Server.

Danach wurden XRIs auf dem alphaServer angelegt. @alpha*institutions ist dazu da, um eine Liste von Institutionen zu verwalten. Wie man sieht, wurde zu @alpha *institutions hinzugefügt. institutions ist also ein dazugekommenes Subsegment, das durch * getrennt ist, was bei i-names üblich ist. Dabei wurde eine neue Instanz erzeugt. Zu dieser Instanz gehört auch ein eigenes XDI-Endpoint, mit dem man kommuniziert, falls man mit XDI-Daten arbeitet. Die Daten werden in einem XDI-Dokument abgelegt.

Da der weitere Aufbau bei allen Institutionen gleich ist, beschränke ich mich hier auf die Beschreibung einer Institution.

Es wurde eine weitere XRI @alpha*institutions*officeprofalice angelegt. Bei dieser wird nur die Information hinterlassen, wo der Server dieser Institution zu erreichen ist. Und anschließend wird der weitere Aufbau von Subsegmenten, Instanzen und beliebigen Informationen auf den Institutionsserver von officeprofalice verlagert. Die weiteren Einzelheiten zu dieser Infrastruktur und dazu, wie die Informationen verteilt sind, liefert der Anhang zu dieser Arbeit.

Auf dem alphaServer wurden außerdem noch weitere optionale XRIs angelegt: @alpha*stdcatalogs und @alpha*stdcatalogs*rolecatalog. Die erste enthält die Liste von

möglichen Katalogen und die zweite einen konkreten Katalog mit den möglichen Rollen von Aktoren in Institutionen.

Bild 6.1 als Screenshot zeigt einen Ausschnitt von dem so entstandenen XRI-Graphen. Der Graph kann beliebig weiter in die Tiefe wachsen. Wie man im Bild sieht und es in den Grundlagen gezeigt wurde, kann ein Subsegment, das zu einer bestimmten Instanz führt, ein i-name, eine i-number und, was neu ist, ein Synonym sein. XRI erlaubt Synonyme. So führt ein Synonym i von institutions ebenfalls zu der gleichen Instanz.

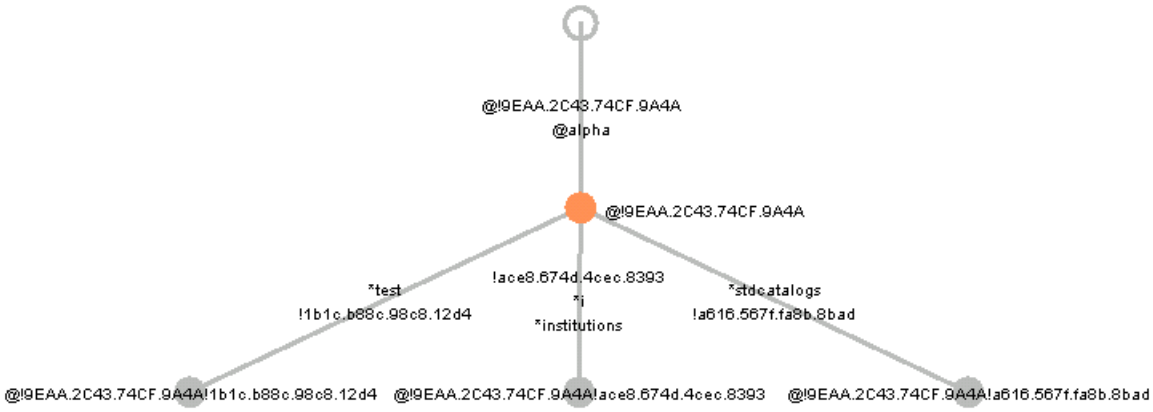


Bild 6.1: XRI Graph

6.2 Client

ibrokerKit verwendet die von Higgins [Hig10a] entwickelte und XDI implementierende Java-Bibliothek [Hig10b]. Diese wurde auch bei der Implementierung vom Client verwendet. Es gibt auf [Hig10c] Tutorials, die beispielhaft die Benutzung dieser Bibliothek erläutern.

Die Grundidee dabei ist die Arbeit mit einem XDI-Graphen, der die Struktur eines XDI-Dokumentes darstellt. Ein XDI-Graph besteht möglicherweise aus: Subjekten, Prädikaten, Literalen, Referenzen und inneren Graphen. Im darauffolgenden Beispiel wird es verdeutlicht. Im XDI-Dokument zu der XRI @alpha*stdcatalogs*rolecatalog sind mögliche Rollen von Aktoren hinterlegt. Das zeigt Listing 6.1.

```
@alpha*stdcatalogs*rolecatalog
    +roles
        radiologist
        oncologist
        gynecologist
```

Listing 6.1: Rollen.

Subjekt ist hier @alpha*stdcatalogs*rolecatalog. +roles ist ein Prädikat. Und die restlichen Einträge sind Literale. Durch die XDI-Adresse @alpha*stdcatalogs*rolecatalog/+roles sind dann diese Literale, in unserem Beispiel die Liste von Rollen, auffindbar. @alpha*stdcatalogs*rolecatalog/+roles ist eine XDI-Adresse in XRI 3.0 Syntax. Wichtig ist hier, dass man mithilfe von der XDI4J-Bibliothek diese Elemente eintragen, löschen und modifizieren kann.

An dieser Stelle ist anzumerken, dass die XRI, die in diesem Fall ein Teil der XDI-Adresse ist (nämlich @alpha*stdcatalogs*rolecatalog), gleichen Namen wie der Subjekt in diesem XDI-Dokument hat. Es ist so implementiert.

Die Vorgehensweise wird an einem Beispiel erklärt.

1. Man will die Liste von allen Rollen erhalten.
2. So verwendet man die Methode getAll() von der Klasse AlphaQClient. AlphaQClient hat außerdem eine Methode, die keine Parameter erwartet und gleiche Ergebnisse liefert. Dies ist möglich, da die XDI-Adresse bekannt ist und innerhalb der Methode verwendet wird. Allerdings verwendet diese Methode selbst die universelle Methode getAll(String iname).
3. Die Liste ist, wie im Listing 6.3 gezeigt, abgespeichert.

```
    @alpha*stdcatalogs*rolecatalog
            +roles
                    radiologist
                    oncologist
                    gynecologist
```

Listing 6.2: Rollen.

4. Man ruft getAll(@alpha*stdcatalogs*rolecatalog/+roles); auf.
5. Es wird eine Nachricht aufgebaut und dem AlphaMessenger übergeben.
6. Dieser muss, bevor er die Nachricht sendet, die Netzwerkadresse von dem entsprechenden XDI-Endpoint ermitteln. Dafür verwendet er AlphaResolver und

übergibt ihm die XDI-Adresse. Der Resolver extrahiert aus der XDI-Adresse (@alpha*stdcatalogs*rolecatalog/+roles) die XRI (@alpha*stdcatalogs*rolecatalog), löst diese auf und liefert dem Messenger die Netzwerkadresse von dem entsprechenden XDI-Endpoint. Der Messenger sendet die Nachricht mit der Anfrage nun an diese Adresse, bekommt die Antwort und gibt diese dem AlphaQClient (der getAll(String iname)-Methode) weiter.

7. AlphaMessenger und AlphaResolver haben je eine Methode, deswegen werden hier zur Vereinfachung nur die Klassennamen verwendet.

6.3 Zusammenfassung

In diesem Kapitel wird die technische Umsetzung von vorgestellten Konzepten mit ausgewählten Ausschnitten vorgestellt. Die Serverseite liefert die Informationen, wie die Infrastruktur aussieht. Auf der Clientseite wird kurz skizziert, wie Clients die XDI-Endpoints auffinden, damit sie später Nachrichten an die Endpoints senden können.

7 Resultate, Bewertung und Ausblick

Die Realisierung zeigt, dass die Verwaltung und Austausch von Daten mittels XRI und XDI durch diese Institutionen durchaus möglich sind. Das einzige Element, das sich die Institutionen dabei teilen, ist der alphaServer. Ansonsten sind sie völlig autonom. Die Realisierung der Aufgabe setzt aber homogene Systeme und einheitliche Struktur beim Anlegen von XRIs voraus. Dies schließt andere Möglichkeiten jedoch nicht aus. Die Skalierbarkeit ist unbegrenzt möglich. Die einfache Haltung vom Client erleichtert dessen Wartung.

Eine andere Frage bei dieser Realisierung ist, ob diese für alle Institutionen im Gesundheitswesen geeignet ist. Dafür muss man die Größe möglicher Institutionen untersuchen. Die kleinste Institution dabei ist eine ärztliche Einzelpraxis, die durchaus aus 3-4 Personen einschließlich des Arztes selbst und ein paar Rechnern bestehen kann. Für eine solche Organisationsform kann diese Art der Verwaltung als zu aufwendig vorkommen. Abhilfe schafft hier die Möglichkeit, die Daten von solchen Institutionen auf dem alphaServer zu verwalten. Auch hier ist es möglich, dass die Institution ihre Daten selbst verwaltet. Generell ist es möglich, dass alle Institutionen ihre Daten auf dem alphaServer selbständig verwalten. In diesem Fall sind die Daten aber nicht mehr verteilt.

8 Zusammenfassung

Diese Arbeit dient als Teil des Projektes „alpha-Flow“. Bei diesem Projekt geht es um aktive Dokumente, die als Unterstützung von Abläufen im Gesundheitswesen gedacht sind. Im Vordergrund dieser Arbeit steht die verteilte Verwaltung von Institutionsdaten durch die Institutionen selbst. Die Kerninformationen sind Institutionen, Aktoren und Rollen von Aktoren in den Institutionen.

Die zu entwickelnde Software muss den Institutionen die Möglichkeit geben ihre Institutionsdaten selbständig zu verwalten. Außerdem muss sie ermöglichen, dass die Institutionen diese Informationen untereinander austauschen können. Dafür hat jede Institution einen Server für die Verwaltung von Daten und einen Client für den Datenaustausch. Der alphaServer dient als zentrale Registratur von Institutionen, damit diese voneinander erfahren können.

Um die Aufgabe zu lösen muss man sich Gedanken darüber machen, was bei einer verteilten Institutionsverwaltung zu beachten ist. Als Erstes muss man sich überlegen, wie man die Institutionen und auch Aktoren identifiziert. Desweiteren ist zu bedenken, wie der Datenaustausch bzw. die Datenverwaltung erfolgt. Diese Probleme werden mithilfe sehr junger Technologien, XRI und XDI, auf die in der Aufgabenstellung als vielversprechend hingewiesen wurde, gelöst. XRI dient in dieser Arbeit der Identifikation und Lokalisierung von Institutions- bzw. Aktoreninstanzen. Mit XDI werden die Verwaltung und der Austausch von Daten realisiert.

Die Serverseite ist mittels eines Softwaresystems namens „ibrokerKit„ aufgebaut. Von den zahlreichen Komponenten von ibrokerKit werden hier solche verwendet, die für die Auflösung von XRIs, die Verwaltung von XRIs und das Arbeiten mit den XDI-Daten, die zu einer bestimmten XRI gehören, dienen. Die Clientseite ermöglicht die administrative Arbeit mit den Institutionsdaten und stellt außerdem Funktionen bereit, die beliebige Suchanfragen ausführen. Die Realisierung zeigt, dass die Verwaltung und Austausch von Daten mittels XRI und XDI durch die Institutionen durchaus möglich sind.

Diese Arbeit zeigt, dass eine verteilte dezentrale Verwaltung von Institutionsinformationen - Metadaten - durch Institutionen selbst möglich ist und wie diese realisiert

werden kann. Diese Lösung könnte man verallgemeinert für alle Arten von Institutionen und nicht nur im Gesundheitswesen verwenden.

Appendices

A Anhang

Dieses Kapitel liefert eine detailliertere Beschreibung, wie das System auf dem Server aufgebaut ist.

A.1 tomcat+mysql

In MySQL müssen folgende DB's mit entsprechenden Benutzern (mit allen Rechten) eingerichtet werden:

```
1    DB:ibroker_openxri; Username:ibroker_openxri; Password:ibroker.
2    DB:ibroker_ibroker; Username:ibroker_ibroker; Password:ibroker.
3    DB:ibroker_iservice; Username:ibroker_iservice; Password:ibroker.
4    DB:ibroker_epptools; Username:ibroker_epptools; Password:ibroker.
```

Listing A.1: MySQL-DBs

Um tomcat mit mysql für ibrokerKit einzurichten, gibt es folgende Abhilfe. Man fügt dem Context-File von tomcat den unten stehenden Eintrag. Dieser Eintrag ist unter /lib-mvn-ext/tomcat+mysql.txt gespeichert. Das Verzeichnis lib-mvn-ext findet man nach dem auschecken des Projekts.

```
1  <Resource
2        name="jdbc/ibroker_openxri"
3        auth="Container"
4        type="javax.sql.DataSource"
5        maxActive="20"
6        maxIdle="10"
7        maxWait="-1"
8        removeAbandoned="true"
9        removeAbandonedTimeout="60"
10       logAbandoned="true"
11       username="ibroker_openxri"
12       password="ibroker"
13       driverClassName="com.mysql.jdbc.Driver"
14       testOnBorrow="true"
```

```
15        testOnReturn="true"
16        testWhileIdle="true"
17        validationQuery="SELECT␣COUNT(*)␣FROM␣subsegment"
18        url="jdbc:mysql://localhost:3306/ibroker_openxri?autoReconnect=
              true" />
19
20  <Resource
21        name="jdbc/ibroker_iservice"
22        auth="Container"
23        type="javax.sql.DataSource"
24        maxActive="20"
25        maxIdle="10"
26        maxWait="-1"
27        removeAbandoned="true"
28        removeAbandonedTimeout="60"
29        logAbandoned="true"
30        username="ibroker_iservice"
31        password="ibroker"
32        driverClassName="com.mysql.jdbc.Driver"
33        testOnBorrow="true"
34        testOnReturn="true"
35        testWhileIdle="true"
36        validationQuery="SELECT␣COUNT(*)␣FROM␣contact"
37        url="jdbc:mysql://localhost:3306/ibroker_iservice?autoReconnect=
              true" />
38
39  <Resource
40        name="jdbc/ibroker_ibroker"
41        auth="Container"
42        type="javax.sql.DataSource"
43        maxActive="20"
44        maxIdle="10"
45        maxWait="-1"
46        removeAbandoned="true"
47        removeAbandonedTimeout="60"
48        logAbandoned="true"
49        username="ibroker_ibroker"
50        password="ibroker"
51        driverClassName="com.mysql.jdbc.Driver"
52        testOnBorrow="true"
53        testOnReturn="true"
54        testWhileIdle="true"
```

```
55        validationQuery="SELECT␣COUNT(*)␣FROM␣user"
56        url="jdbc:mysql://localhost:3306/ibroker_ibroker?autoReconnect=
              true" />
57
58  <Resource
59        name="jdbc/ibroker_epptools"
60        auth="Container"
61        type="javax.sql.DataSource"
62        maxActive="20"
63        maxIdle="10"
64        maxWait="-1"
65        removeAbandoned="true"
66        removeAbandonedTimeout="60"
67        logAbandoned="true"
68        username="ibroker_epptools"
69        password="ibroker"
70        driverClassName="com.mysql.jdbc.Driver"
71        testOnBorrow="true"
72        testOnReturn="true"
73        testWhileIdle="true"
74        validationQuery="SELECT␣COUNT(*)␣FROM␣action"
75        url="jdbc:mysql://localhost:3306/ibroker_epptools?autoReconnect=
              true" />
```

Listing A.2: Context-File-Eintrag von Tomcat.

Dabei bitte beachten - beim ersten Start von ibrokerKit in der validationQuery=“SELECT COUNT(*) .....“ für alle DB’s COUNT(*)... auf 1 ersetzen. Z.B validationQuery=“SELECT COUNT(*) FROM action“ durch validationQuery=“SELECT 1“ ersetzen. Sonst gibt es eine Fehlermeldung. Der Grund ist, dass es vor dem ersten Start des Programms noch keine Tabellen existieren, die man mit der Anweisung überprüft.

A.2 ibrokerKit

Wie schon in der Ausarbeitung beschrieben, wurden die Komponenten von ibrokerKit, die für eigene Projekte modifiziert werden müssen, umbenannt (s. auch die Anleitung zu ibrokerKit). Die Bezeichnung „youribroker“ wurde auf „alpha“ geändert. ibrokerKit hat mehrere Komponenten. Für diese Aufgabe reichen jedoch www.alpha.com, resolve.alpha.com, xdi.alpha.com und admin-openxri.alpha.com.

resolve... ist ein OpenXRI-Server, der verschiedene XRI's auflöst. Dabei geht es um XRI's, die auf dem gleichen Server verwaltet werden, auf dem resolve.alpha.com auch läuft. Wenn z.B. eine Instanz einer bestimmten XRI auf diesem Server verwaltet wird, dann ist dieser Server für deren Auflösung zuständig. So kann ein Client bei resolve.alpha.com die XDI-Endpoint-Adresse nachfragen, die bei der entsprechenden Instanz hinterlegt ist.

Man muss in web.xml von OpenXRI-Server die URI eintragen, unter der der Server auch läuft. Das ist wichtig, weil das System die URIs in einigen Fällen automatisch einträgt. Dazu später.

admin-openxri.alpha.com ist ein Administration-Interface für die Verwaltung von OpenXRI-DB.

xdi.alpha.com ist ein XDI-Frontend. Im Browser nach dem Start sieht man eine leere Seite. Das muss auch so sein. Nur wenn man die XDI-Endpoint-Adresse einer bestimmten XRI im Browser eingibt, sieht man die entsprechenden Einträge.

www.alpha.com ist die eigentliche Benutzerschnittstelle. Hier registrieren die Benutzer Ihre XRI's. XRI's sind entweder i-names oder i-numbers. Zur einheitlichen Darstellung wird hier die Bezeichnung XRI verwendet.

Mit jeder XRI sind verschiedene Dienste verknüpft. Diese sind in einem XRD-Dokument hinterlegt. Mögliche Dienste sind Contact, Forwarding,..... Für diese Aufgabe sind jedoch Resolution- und XDI-Dienste von Bedeutung. Formal sind diese Dienste in den jeweiligen sogenannten SEP's beschrieben. SEP bedeutet Service End Point. Bei einem Resolve-SEP einer XRI ist die URI eingetragen, unter der die Sub-XRI's zu finden sind. Im XDI-SEP ist die URI eingetragen, unter der der entsprechende XDI-Endpoint läuft.

Zwei Beispiele sollen dies verdeutlichen:

Resolve-SEP: für diese Aufgabe wurde die XRI namens @alpha bei www.fullxri.com registriert. Die Sub-XRI's von @alpha (z.B. @alpha*institutions) werden auf dem „alphaServer“ verwaltet. So sieht dann der ensprechende Resolve-SEP von @alpha aus:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Service priority="10" xmlns="xri://$xrd*($v*2.0)">
3 <ProviderID>xri://@alpha</ProviderID>
4 <Type select="true">xri://$res*auth*($v*2.0)</Type>
5 <MediaType select="true">application/xrds+xml</MediaType>
6 <URI append="none" priority="2">
7 http://131.188.36.135:8080/com.alpha.resolve-0.5/ns/@alpha/</URI>
8 <grsid xmlns="">sep-281151896</grsid>
9 </Service>
```

Listing A.3: Resolve-SEP von @alpha.

Wichtig ist hier die URI, die man manuell eintragen muss, da in diesem Fall die Sub-XRI's auf einem anderen Server verwaltet werden. Falls man Sub-XRIs auf dem gleichen Server anlegen würde, hätte das System diese URI automatisch eingetragen.

XDI-SEP: die URI dafür muss man im Gegensatz zu Resolve immer per Hand eintragen. So ist es zumindest im ibroker-System. Wenn der XDI-Frontend unter http://131.188.36.135:8080/com.alpha.xdi-0.5/ läuft, dann ist die Uri vom XDI-Endpoint von @alpha*institutions http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*institutions/.

Die Komponente www.alpha.com liefert beim Registrieren von XRI's die Vorlagen für entsprechende SEP's. Die Anleitung zu ibrokerKit beschreibt, wie man mit solchen SEP's arbeitet.

An dieser Stelle ist zu erwähnen, dass ibrokerKit die XDI-Dokumente in einer BDB verwaltet. Diese befindet sich im bin-Verzeichnis von tomcat. Genauer gesagt in einem Unterverzeichnis von bin. In unserem Fall ist es Unterverzeichnis xdiFront-xdi.alpha.com .

A.3 Installation/Start

In diesem Abschnitt werden die einzelnen Schritte der Installation beschrieben. Das Projekt liegt unter
svn+ssh://siigenge@faui6svn.informatik.uni-erlangen.de/proj/svn/projekte/promed/src-

/alphainstitutions/trunk.

1. Nach dem Auschecken des Projektes in das Arbeitsverzeichnis findet man folgende Unterverzeichnisse: openxri, ibrokerkit, promed-institutions und lib-mvn-ext.
2. Mit „cd lib-mvn-ext“ wechselt man in das Verzeichnis „lib-mvn-ext“ und führt die Datei install.sh (Unix) bzw. install.cmd (Windows) aus. Damit werden vom maven die externen Bibliotheken installiert, die nicht automatisch installiert werden können. Falls es dabei Probleme geben sollte, sind die Befehle zur Installation unter /lib-mvn-ext/mvn-install.txt gespeichert. Außerdem sind diese Einträge im Anhang unter A.7 aufgeführt.
3. Dann wechselt man mit „cd ..“ und „cd openxri“ in das openxri-Verzeichnis und gibt „mvn install“ ein.

4. Danach wechselt man mit „cd ..“ und „cd promed-institutions“ in das promed-institutions-Verzeichnis und gibt „mvn install“ ein. Bei diesem Punkt wird der Client installiert. Den Client kann man bereits verwenden ohne weitere Schritte auszuführen, da das Testsystem, wie in diesem Anhang beschrieben, bereits eingerichtet wurde. Weitere Schritte werden benötigt, falls man das ganze System komplett neu einrichten will.
5. Dann wechselt man mit „cd ..“ und „cd ibrokerkit/ibrokerKit“ in das ibrokerkit/ibrokerKit-Verzeichnis und gibt „mvn install“ ein.
6. Nach diesen Ausführungen werden die war-files in den entsprechenden Verzeichnissen erstellt.
7. Notwendige Komponenten werden nun gestartet. Das sind:
   /ibrokerkit/com.alpha.resolve/target/com.alpha.resolve-0.5.war;
   /ibrokerkit/com.alpha.www/target/com.alpha.www-0.5.war;
   /ibrokerkit/com.alpha.xdi/target/com.alpha.xdi-0.5.war und
   /ibrokerkit/com.alpha.admin-openxri/target/com.alpha.admin-openxri-0.5.war.
   Diese Komponenten können mit dem Tomcat Manager deployed und gestartet werden.
8. @alpha bei www.fullxri.com registriert.
   Dabei http://131.188.36.135:8080/com.alpha.resolve-0.5/ns/@alpha/ im Res-SEP von @alpha eingetragen. Das musste man nur einmal durchführen. Falls man eine andere Root-XRI registrieren will, kann man dies entweder unter www.fullxri.com oder http://www.freexri.com tun.
9. @alpha als Root-Authority auf dem alphaServer über admin-openxri eingetragen. Dabei der @alpha unbedingt i-number im XRD vergeben - die gleiche i-number, die auf www.fullxri.com bei der Registrierung von @alpha vergeben wurde. Sonst kein XDI möglich. (das über admin-openxri.alpha.com)
10. Die benötigten XRI's angelegt. Zu jeder auch XDI-Endpoint-URI. (alles über www.alpha.com)
11. Nachdem alle Instanzen angelegt sind, kann man mit den Clients arbeiten.

A.4 XRI-Infrastruktur

Hier ist die Liste von angelegten XRIs und die URIs von laufenden Komponenten auf dem alphaServer. Auf dem alphaServer wurden zu Testzwecken auch die Institutions-XRIs angelegt. Es würde aber genauso funktionieren, wenn man diese auf die jeweiligen Institutionsserver verlagert.

URIs von Komponenten:

- http://131.188.36.135:8080/com.alpha.xdi-0.5/ . Root XDI-Endpoint ohne eine konkrete XRI.
- http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*stdcatalogs*rolecatalog/. XDI-Endpoint für @alpha*stdcatalogs*rolecatalog.
- http://131.188.36.135:8080/com.alpha.www-0.5/ . User-Endpoint für die Verwaltung von XRIs. Einloggen mit @alpha*institutions. Darunter erreicht man auch Subsegmente, wie z.B. @alpha*institutions*officeprofalice. Oder einloggen mit @alpha*stdcatalogs. Das Passwort ist immer „promed“.
- http://131.188.36.135:8080/com.alpha.resolve-0.5/. Komponente, die für die Auflösung zuständig ist.
- http://131.188.36.135:8080/com.alpha.admin-openxri-0.5/home/Index/. Administration von openXRI. Eigentlich optional, man kommt aber ohne diese Komponente nicht aus. Wichtig für das Anlegen von der Root-XRI.

XRIs:

```
1  @alpha
2  @alpha*stdcatalogs
3  @alpha*stdcatalogs*rolecatalog
4  @alpha*institutions
5  @alpha*institutions*officeprofalice
6  @alpha*institutions*officeprofalice*idesc
7  Beschreibung der Institution.
8  Zum Beispiel Liste der Rollen mit ensprechenden Aktoren.
9
10 @alpha*institutions*officeprofalice*actors
11 Liste der Aktoren in officeprofalice.
12 @alpha*institutions*officeprofalice*actors*profalice
13 Konkreter Aktor.
```

Listing A.4: Angelegte XRIs.

Ähnlich wie officeprofalice sind auch officedrbob und ucc-asklepios aufgebaut. Verteilung von XRIs:

```
alphaServer:
@alpha - Root (in der Sprache von openXRI - Root-Authority)

@alpha*stdcatalogs
@alpha*stdcatalogs*rolecatalog

@alpha*institutions
@alpha*institutions*officeprofalice
@alpha*institutions*officedrbob
@alpha*institutions*ucc-asklepios

Danach auf dem jeweiligen Institutionsserver(hier officeprofalice):

@alpha*institutions*officeprofalice - Root

@alpha*institutions*officeprofalice*idesc
Beschreibung der Institution.
Zum Beispiel Liste der Rollen mit ensprechenden Aktoren.
@alpha*institutions*officeprofalice*actors
Liste der Aktoren.
@alpha*institutions*officeprofalice*actors*profalice
Konkreter Aktor.
```

Listing A.5: Verteilung von XRIs.

A.5 Beispiel

Falls man sehen möchte was in einem XDI-Dokument gespeichert ist und wie dieser aufgebaut ist, so kann man die URI von einem XDI-Endpoint im Browser eingeben. Bei der Eingabe von http://131.188.36.135:8080/com.alpha.xdi-0.5/@alpha*institutions/ sieht man folgendes:

```
$
    $http$uri$1
        "http://131.188.36.135:8080/com.alpha.xdi-0.5/@!9EAA.2C43.74CF
           .9A4A!ace8.674d.4cec.8393/"
    $is($xdi$v$1)
        @!9EAA.2C43.74CF.9A4A!ace8.674d.4cec.8393
    $is$a
```

```
7         ($xdi$v$1)
8         ($pds$v$1)
9 @alpha*institutions
10     +institutions
11         @alpha*institutions*officeprofalice
12         @alpha*i*ucc-asklepios
13         @alpha*institutions*officedrbob
```

Listing A.6: XDI-Endpoint von @alpha*institutions.

A.6 Entwicklungswerkzeuge

Liste von verwendeten Werkzeugen:

- Eclipse Helios SR1
- Apache Maven 3.0
- Apache Tomcat 6.0.29
- MySQL Server 5.1

A.7 Aushilfe bei Problemen mit install.sh bzw. install.cmd

```
1 mvn install:install-file  -Dfile=xdi4j-1.0.700.jar  -DgroupId=org.
    eclipse.higgins -DartifactId=xdi4j -Dversion=1.0.700 -Dpackaging=
    jar
2 mvn install:install-file  -Dfile=epp-0.4.9.jar  -DgroupId=com.neulevel
     -DartifactId=epp -Dversion=0.4.9 -Dpackaging=jar
3 mvn install:install-file  -Dfile=openid4java-nodeps-0.9.5-SNAPSHOT.jar
      -DgroupId=org.openid4java -DartifactId=openid4java-nodeps -
    Dversion=0.9.5-SNAPSHOT -Dpackaging=jar
4 mvn install:install-file  -Dfile=je-4.0.103.jar  -DgroupId=com.
    sleepycat -DartifactId=je -Dversion=4.0.103 -Dpackaging=jar
5 mvn install:install-file  -Dfile=mail-1.4.2.jar  -DgroupId=javax.mail
    -DartifactId=mail -Dversion=1.4.2 -Dpackaging=jar
```

Literaturverzeichnis

[Chr09] Christoph P. Neumann and Richard Lenz, editor. *alpha-Flow: A Document-based Approach to Inter-Institutional Process Support in Healthcare*, September 2009.

[Dru04] Drummond Reed and Geoffrey Strongin. The Dataweb: An Introduction to XDI. A White Paper for the OASIS XDI Technical Committee v2. `http://www.oasis-open.org/committees/download.php/6434/wd-xdi-intro-white-paper-2004-04-12.pdf`, April 12 2004.

[Dru05] Drummond Reed and Dave McAlpin. Extensible Resource Identifier (XRI) Syntax V2.0 Committee Specification. `http://www.oasis-open.org/committees/download.php/15376/xri-syntax-V2.0-cs.html`, 14 November 2005.

[Gab08] Gabe Wachob and Drummond Reed and Les Chasen and William Tan and Steve Churchill. Extensible Resource Identifier (XRI) Resolution Version 2.0 Committee Specification 01. `http://docs.oasis-open.org/xri/xri-resolution/2.0/specs/cs01/xri-resolution-V2.0-cs-01.html`, 12 April 2008.

[Hig10a] Higgins Project. XDI Implementierung. `http://download.eclipse.org/technology/higgins/downloads/xdi4j/builds/S-S20090822-200908220210/javadoc/overview-summary.html`, 2010.

[Hig10b] Higgins Project. XDI4j. `http://wiki.eclipse.org/XDI4j`, 2010.

[Hig10c] Higgins Project. XDI4j Tutorials. `http://wiki.eclipse.org/XDI4j_Tutorials`, 2010.

[Len09] Richard Lenz. Information Systems in Healthcare– state and steps towards sustainability. IMIA Yearbook of Medical Informatics 2009 (2009), S. 63-70, 2009. - A Position Paper -.

[Mic05] Michael Franklin and Alon Halevy and David Maier, editor. *From databases to dataspaces: a new abstraction for information management*. ACM New York, NY, USA, 2005.

[OAS04] OASIS XRI Data Interchange (XDI) TC. The XDI FAQ. `http://www.oasis-open.org/committees/xdi/faq.php`, January 20 2004.

[OAS08] OASIS Extensible Resource Identifier TC. XRI 2.0 FAQ. `http://www.oasis-open.org/committees/xri/faq.php`, May 2008.

[Ree08] Drummond Reed. XRI in a Nutshell. `http://www.equalsdrummond.name/?p=150`, September 2008.

[Tet02] Tetsuo TAMAI, editor. *Evolvable Programming based on Collaboration-Field and Role Model*. ACM New York, NY, USA ©2002, 2002.

[Tet05] Tetsuo Tamai and Naoyasu Ubayashi Kyushu and Ryoichi Ichiyama, editor. *An Adaptive Object Model with Dynamic Role Binding*. ACM New York, NY, USA ©2005, 2005.