Entwurf und Realisierung eines IHE XDS-Komponententeststands

Studienarbeit im Fach Informatik

vorgelegt von

Florian Wagner

geb. am 25. Juli 1983 in Lauf a. d. Pegnitz

angefertigt am

Institut für Informatik
Lehrstuhl für Informatik 6 – Data Management
Friedrich-Alexander-Universität Erlangen–Nürnberg
(Prof. Dr. K. Meyer-Wegener)

Betreuer: Prof. Dr. Richard Lenz
Dipl.-Inf. Christoph Neumann

Beginn der Arbeit: 15. April 2008
Abgabe der Arbeit: 15. Januar 2009

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die 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 die Informatik 6 (Data Management), 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 15. Januar 2009

Florian Wagner

Inhaltsverzeichnis

1. Einleitung

Ein gemeinsamer Zugriff auf medizinische Dokumente im Zuge der Behandlung eines Patienten findet heute beinahe ausschließlich nur innerhalb der Krankenhäuser statt. Zwischen Krankenhäusern bzw. niedergelassenen Ärzten und Krankenhäusern beschränkt sich der Austausch von Informationen auf den obligatorischen Arztbrief nach erfolgter Behandlung. Erstrebenswert wäre aber ein kurzfristiger Zugriff auf die medizinische Historie eines Patienten, damit zum Beispiel bei einer Einlieferung in die Notaufnahme dort sofort Informationen wie etwaige Vorerkrankungen, kürzlich aufgetretene Beschwerden oder vor kurzem von einem niedergelassenen Arzt verschriebene Medikamente abgerufen werden können. Voraussetzung hierfür ist jedoch sowohl die systematische digitale Erfassung von Patientendaten, als auch eine standardisierte Schnittstelle zum Austausch der Daten zwischen den einzelnen Institutionen. Ersteres ist gerade bei niedergelassenen Ärzten auch wegen der hohen Investitionskosten in entsprechende IT-Systeme noch im Aufbau, letzteres ist notwendig, da sich oftmals selbst innerhalb einzelner Institutionen die IT-Landschaft sehr heterogen gestaltet, von der Heterogenität zwischen Institutionen ganz zu schweigen.

Ein Ansatz zum digitalen Austausch medizinischer Dokumente ist das „Cross Enterprise Document Sharing“-Profil (XDS) der IHE. Es beschreibt ein verteiltes System aus mehreren Komponenten, wobei eine Komponente als zentrale Speicherungsinstanz dient. Für dieses Profil existiert eine Open-Source-Implementierung des „Open Healthcare Framework“-Projekts (OHF), an dem unter anderem IBM und die Mayo Clinic beteiligt sind.

Ziel dieser Arbeit ist es ein einfaches XDS-System zu realisieren, mit dem sich verschiedene XDS-Komponenten-Implementierungen auf ihre Konformität, Stabilität und Funktionalität überprüfen lassen. Im Rahmen dieser Arbeit soll ebenfalls die XDS-Implementierung der OHF auf Umfang, Stabilität und Tragfähigkeit geprüft werden.

Die vorliegende Arbeit gliedert im wesentlichen in sechs Abschnitte. Zuerst erfolgt ein kurzer Überblick über die verschiedenen Technologien und Standards, die im Rahmen von XDS zum Einsatz kommen, allen voran natürlich das XDS-Profil selbst. Daran schließt sich ein Kapitel an, in dem einerseits die Anforderungen an ein System gemäß dem XDS-Profil, andererseits aber auch die speziellen Anforderungen der Testumgebung dokumentiert werden. Das folgende Kapitel „Stand der Technik“ befasst sich mit dem erwähnten OHF und vergleicht diesen mit den zuvor gesammelten Anforderungen. Anschließend erfolgt eine grobe Übersicht über die erarbeitete Architektur des Gesamtsystems und die Diskussion einiger Entwurfsalternativen für das Serversubsystem. Das

folgende Kapitel widmet sich dann dem Systementwurf im Detail. Zuletzt erfolgt ein kritischer Vergleich des entstandenen Systems mit den zusammengetragenen Anforderungen und eine Sammlung noch offener Punkte.

2. Grundlagen

2.1. Cross-Enterprise Document Sharing (XDS)

2.1.1. Überblick

Das Cross-Enterprise-Document-Sharing-Profil (kurz *XDS*) ermöglicht es mehreren medizinischen Versorgungseinrichtungen medizinische Patientendaten in Form von Dokumenten zentral zu speichern und so allen Teilnehmern zur Verfügung zu stellen. XDS wurde als Profil durch die IHE, einer Initiative von Medizinern und Wirtschaft mit dem Ziel, die Zusammenarbeit zwischen Computersystemen im medizinischen Umfeld zu verbessern, standardisiert.

Der Standard definiert vier Aktoren, die mit Hilfe von vier Transaktionen miteinander interagieren können (Abbildung 2.1). Durch den Document-Source-Aktor werden Dokumente in das System eingestellt, indem sie und zusätzliche Metadaten an den Document-Repository-Aktor gesendet werden, der für die persistente Speicherung zuständig ist. Dieser registriert die erhaltenen Dokumente mit Hilfe der Metadaten am Document-Registry-Aktor. Suchanfragen nach Dokumenten werden über den Document-Consumer-Aktor ebenfalls an den Document-Registry-Aktor gestellt, welcher die gespeicherten Metadaten sowie die Adresse des Dokuments im Document-Repository-Aktor zurückliefert. Mit Hilfe der Adresse kann dann das Dokument in einem weiteren Schritt vom Document-Consumer-Aktor am Document-RepositoryAktor abgerufen werden. Die Verwendung von Metadaten ist notwendig, da XDS als inhaltneutrales System ausgelegt ist und deshalb keine Informationen aus den eigentlichen Dokumenten extrahieren kann. Es ist dadurch aber auch möglich, neben reinen Text- oder XML-formatierten Dokumenten (z.B. HL7 CDA) Bilder (z.B. DICOM) oder PDF-Dateien über XDS zu verwalten.

Alle Teilnehmer an einem XDS-System müssen der gleichen „Affinity Domain“ angehören. „Affinity Domain“ bezeichnet in diesem Zusammenhang eine Gruppe von medizinischen Versorgungseinrichtungen, die auf der Basis von gemeinsamen Richtlinien zusammenarbeiten und sich gegebenenfalls auch Infrastruktur teilen. Hierunter versteht man unter Anderem:

- Einen Krankenhausverbund, d.h. den Zusammenschluss mehrerer Krankenhäuser in einer Region.
- Mehrere Abteilungen gleicher Spezialisierung bzw. Fachrichtung innerhalb eines oder mehrere Krankenhäuser.

- Den Verbund eines oder mehrerer Krankenhäuser mit niedergelassenen Ärzten innerhalb einer Region.

Eine „Affinity Domain“ zeichnet sich vor allem durch gemeinsame Richtlinien für die Identifikation von Patienten — z.B. mit Hilfe einer eindeutigen Identifikationsnummer —, für Zugangsberechtigungen zu den Patientendaten und über die Struktur und das Format der auszutauschenden Daten aus. Hier definiert der XDS-Standard keine Vorgaben, lässt sich aber entsprechend flexibel anpassen.

Technisch basiert XDS auf einer Reihe verbreiteter Standards, darunter ebXML, SOAP und HTTP. Der XDS-Registry-Aktor implementiert eine Registry im Sinne der „ebXML Registry Services“, weshalb XDS auch konsequent ein ebXML-RIM-basiertes Datenmodell und ebXML-codierte Nachrichten, die via SOAP und HTTP versendet werden, zur Interaktorkommunikation einsetzt. Neben der Kombination aus SOAP und HTTP sieht die Spezifikation alternativ auch die Kommunikation der Aktoren via SMTP vor. Dies ist speziell für den Fall gedacht, dass nicht alle an einer Transaktion beteiligten Aktoren gleichzeitig verfügbar sind und somit ein „store and forward“-Mechanismus zur Nachrichtenübermittlung eingesetzt werden muss.

Unter `http://ihewiki.wustl.edu/wiki/index.php/XDS_Main_Page` wird ein Wiki betrieben, welches zusätzlich zu den Spezifikationen (bei IHE „Technical Frameworks“ genannt) nützliche Informationen zur Entwicklung und zum Betrieb von XDS-Komponenten enthält. Dort wird auch ein Test-Client-Programm angeboten, welches es mit ebenfalls angebotenen Testfällen erlaubt, eine XDS-Server-Implementierung auf ihre Funktionsfähigkeit hin zu überprüfen. Damit Entwickler auch direkt die Zusammenarbeit ihrer XDS-Komponenten mit denen anderer Hersteller überprüfen können, findet jährlich in den USA und Europa jeweils ein sogenannter *Connectathon* statt.

2.1.2. Klassen

XDS definiert auf Basis von ebXML RIM eine Reihe von Klassen (Tabelle 2.1), die sowohl in Transaktionen, als auch in Aktoren Verwendung finden. Wichtig sind sie jedoch speziell zum Verständnis des ebXML-RS-basierten Übertragungsprotokolls der einzelnen Transaktionen und dem Persistieren der Metadaten im Document-Registry-Aktor. Zu jeder Klasse werden außerdem die jeweiligen Klassenattribute in tabellarischer Form dargestellt. Hierbei gelten jeweils für die Spalte *Flag* die in Tabelle 2.2 aufgeführten Abkürzungen. Tabelle 2.3 beschreibt außerdem häufig verwendete Datentypen in XDS.

XDSDocumentEntry Objekte dieser Klasse repräsentieren ein Dokument im Sinne von XDS, d.h. die Struktur enthält ausschließlich Metadaten. Das eigentliche Dokument wird abhängig von der jeweiligen Transaktion im MIME-Teil der Anfrage übertragen bzw. durch einen Verweis, der den Speicherort des Dokuments in einem Document-Repository-Aktor angibt, referenziert. Jedes XDSDocumentEntry-Objekt ist genau ei-

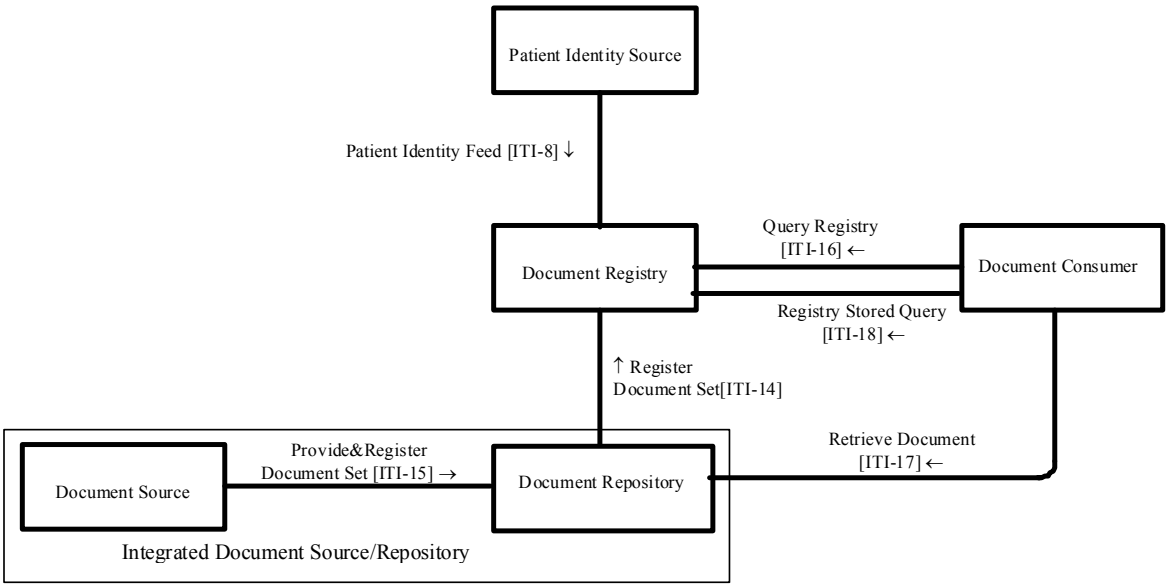

Abbildung 2.1.: XDS-Aktoren und -Transaktionen (aus [iti08a])

XDS-Objektklasse	ebXML-RIM-Oberklasse	mögliche Kindobjektklassen
XDSDocumentEntry	ExtrinsicObject	–
XDSFolder	RegistryPackage	XDSDocumentEntry
XDSSubmissionSet	RegistryPackage	XDSDocumentEntry, XDSFolder, Association
–	Association	–

Tabelle 2.1.: XDS-Klassen

Abkürzung	Bedeutung
C	Der Wert wird von einem der beteiligten Aktoren automatisch festgelegt.
R	Der Wert muss vom Benutzer angegeben werden.
O	Die Angabe des Wertes ist optional.

Tabelle 2.2.: Bedeutung der Spalte *Flag*

nem Patienten zugeordnet. Tabelle 2.4 zeigt eine Auswahl von XDSDocumentEntry-Attributen, darunter alle, die beim Anlegen des Eintrags vorhanden sein müssen.

Datentyp	Format	Beschreibung
CX	<ID>^^^&<OIDderAD>&ISO	Dieser dem HL7-v2-Standard entnommene Datentyp kennzeichnet einen Patienten eindeutig. Er besteht aus einer in der Affinity-Domain eindeutigen ID und der OID der Affinity-Domain.
DTM	JJJJ[MM[DD[hh[mm[ss]]]]]	Datumsangabe.
OID	–	ISO-Objektidentifikator von maximal 64 Zeichen Länge, bestehend aus den Ziffern 0 bis 9 und dem Zeichen „.“ (Punkt).
URI	–	Uniform Resource Identifier gemäß RFC2616
UUID	urn:uuid:<hexadez. UUID-String>	Universally Unique Identifier nach URN-Syntax

Tabelle 2.3.: Häufig verwendete Datentypen

Name	Flag	Datentyp	Bemerkung
author	O	zusammengesetzt	Mögliche Subattribute: authorInstitution, authorPerson, authorRole, authorSpeciality
availabilityStatus	C		Bei erfolgreichem Anlegen wird der Wert ‘Approved’ durch den Registry-Aktor zugewiesen.
classCode	R		Werte Affinity-Domain-spezifisch
classCodeDisplayName	R		Werte Affinity-Domain-spezifisch
comments	O		
confidentialityCode	R		Werte Affinity-Domain-spezifisch
creationTime	R	DTM	
entryUUID	C	UUID	durch Source-, Registry- oder Repository-Aktor zugewiesener interner, technischer Schlüssel

Tabelle 2.4 — Fortsetzung von vorheriger Seite

Name	Flag	Datentyp	Bemerkung
formatCode	R		Werte Affinity-Domain-spezifisch
hash	C	SHA1-Hash	wird durch den Repository-Aktor errechnet
healthcareFacility-TypeCode	R		Werte Affinity-Domain-spezifisch
healthcareFacility-TypeCodeDisplayName	R		Werte Affinity-Domain-spezifisch
languageCode	R		
mimeType	R		
patientId	R	CX	
practiceSettingCode	R		Werte Affinity-Domain-spezifisch
practiceSettingCode-DisplayName	R		Werte Affinity-Domain-spezifisch
serviceStartTime	R	DTM	
serviceStopTime	R	DTM	
size	C	Integer	wird durch den Repository-Aktor berechnet
sourcePatientId	R	CX	Patienten-Id aus der Domäne des Document-Source-Aktors
sourcePatientInfo	R		demographische Informationen über den Patienten
title	O		
typeCode	R		Werte Affinity-Domain-spezifisch
typeCodeDisplayName	R		Werte Affinity-Domain-spezifisch
uniqueId	R	OID	externer, global eindeutiger Schlüssel
URI	C	URI	Adresse, von der das Dokument bezogen werden kann

Tabelle 2.4.: Auswahl von XDSDocumentEntry-Attributen

XDSFolder Objekte dieses Typs repräsentieren einen Ordner zur Strukturierung einer Menge an Dokumenten. XDS erlaubt durch Ordner jedoch nur die Bildung einer flachen Hierarchie, sodass ein Ordner keine Unterordner enthalten darf. Die Spezifikation merkt

jedoch an, dass in zukünftigen Versionen möglicherweise eine tiefe Hierarchie erlaubt wird. Es ist außerdem zu beachten, dass jedes XDSFolder-Objekt genau einem Patienten zugeordnet ist und alle enthaltenen XDSDocumentEntry-Objekte dem gleichen Patienten zugeordnet sein müssen!

Name	Flag	Datentyp	Bemerkung
availabilityStatus	C		Bei erfolgreichem Anlegen wird der Wert 'Approved' durch den Registry-Aktor zugewiesen.
codeList	R	mehrwertig	Werte Affinity-Domain-spezifisch
codeDisplayNameList	R	mehrwertig	Werte Affinity-Domain-spezifisch
comments	O		
entryUUID	C	UUID	durch Source-, Registry- oder Repository-Aktor zugewiesener interner, technischer Schlüssel
lastUpdateTime	C	DTM	wird durch den Registry-Aktor zugewiesen
patientId	R	CX	
title	O		
uniqueId	R	OID	externer, global eindeutiger Schlüssel

Tabelle 2.5.: XDSFolder-Attribute

XDSSubmissionSet Objekte der Submission-Set-Klasse assoziieren die Menge an Dokumenten und Ordnern, die in einem atomaren Vorgang in ein XDS-System eingebracht werden. Ein Submission-Set-Objekt darf auch bereits in einem Document-Registry-Aktor vorhandene Dokumente bzw. Ordner referenzieren und darf ebenso mehrere Dokumente enthalten, die jeweils für unterschiedliche Ordner bestimmt sind oder bereits in ihnen enthalten sind. Einem Übertragungsvorgang ist jeweils genau ein XDSSubmissionSet-Objekt zugeordnet, das bedeutet, dass das gleiche XDSSubmissionSet-Objekt nicht über mehrere Übertragungsvorgänge hinweg eingesetzt werden kann. Genauso darf in einem Übertragungsvorgang nur genau ein XDSSubmissionSet-Objekt enthalten sein. Des Weiteren ist anzumerken, dass ein XDSSubmissionSet-Objekt genau einem Patienten zugeordnet ist und deshalb alle assoziierten Objekte ebenfalls diesem Patienten zugeordnet sein müssen.

Association Dient der Assoziation von Dokumenten zueinander zu Zwecken der Versionierung und digitalen Signierung bzw. von Dokumenten zu Ordnern und Dokumenten zu Submission-Sets.

Zum Beispiel im Falle der Versionierung gibt es die Möglichkeit, ein bereits dem Registry-Aktor bekanntes XDSDocumentEntry-Objekt (targetObject) durch ein im Submission-

Name	Flag	Datentyp	Bemerkung
author	O	zusammen-gesetzt	Mögliche Subattribute: authorInstitution, authorPerson, authorRole, authorSpeciality
availabilityStatus	C		Bei erfolgreichem Anlegen wird der Wert 'Approved' durch den Registry-Aktor zugewiesen.
comments	O		
contentTypeCode	R		Werte Affinity-Domain-spezifisch
contentTypeCodeDisplayName	R		Werte Affinity-Domain-spezifisch
entryUUID	C	UUID	durch Source-, Registry- oder Repository-Aktor zugewiesener interner, technischer Schlüssel
patientId	R	CX	
sourceId	R	OID	Identifikator der Instanz des Document-Source-Aktors
submissionTime	R	DTM	Zeitpunkt der Erstellung des XDSSubmissionSet-Objekts durch den Source-Aktor
title	O		
uniqueId	R	OID	externer, global eindeutiger Schlüssel

Tabelle 2.6.: XDSSubmissionSet-Attribute

Name	Flag	Datentyp	Bemerkung
targetObject	R	UUID	Das Zielobjekt der Assoziation, abhängig vom Typ der Assoziation.
sourceObject	R	UUID	Das Quellobjekt der Assoziation, abhängig vom Typ der Assoziation
associationType	R		Der Typ der Assoziation.

Tabelle 2.7.: Association-Attribute

Set enthaltenes XDSDocumentEntry-Objekt (sourceObject) zu erweitern (association-

Type APND), zu ersetzen (RPLC) bzw. das neue Dokument als eine Transformation des alten zu kennzeichnen (XFRM bzw. XFRM_RPLC).

2.1.3. Aktoren

Dieser Abschnitt beschreibt die Verantwortlichkeiten der einzelnen Aktoren gemäß [iti08a].

Document Source Der Document-Source-Aktor stellt die im XDS-System zu speichernden Dokumente bereit. Der Aktor sendet diese zur Speicherung an Document-Repository-Aktoren und bereitet auch die für die nachfolgende Registrierung an einem Document-Registry-Aktor nötigen Metadaten vor und liefert diese dem Repository-Aktor mit.

Document Consumer Der Document-Consumer-Aktor sendet Suchanfragen an Document-Registry-Aktoren, um diese nach bestimmten Kriterien entsprechenden Objekten zu durchsuchen. Außerdem kann der Aktor gefundene Dokumente an Hand der von Registry-Aktoren gelieferten Metadaten von Document-Repository-Aktoren abrufen.

Document Registry Der Document-Registry-Aktor speichert Metadaten zu allen Dokumenten, die erfolgreich bei ihm registriert wurden, in Form eines XDSDocumentEntry-Objekts. Dies beinhaltet auch einen Verweis auf den Speicherort des Dokuments in einem Document-Repository-Aktor. Zusätzlich verfügt der Document-Registry-Aktor auch über Informationen zur Strukturierung der Dokumentenmenge durch XDSFolder und XDSSubmissionSets. Vor der Registrierung neuer Objekte werden deren Metadaten vom Aktor auf ihre Gültigkeit und Standardkonformität hin überprüft und gegebenenfalls deren Registrierung abgelehnt. Der Aktor ist ebenso dafür verantwortlich, Suchanfragen von Document-Consumer-Aktoren entgegenzunehmen, seinen Datenbestand nach Objekten, die den geforderten Suchkriterien entsprechen, zu durchsuchen und diese in geeigneter Form an den initiierenden Consumer-Aktor zurückzuliefern.

Document Repository Der Document-Repository-Aktor ist für die persistente Speicherung aller ihm durch Document-Source-Aktoren übergebener Dokumente und deren Registrierung an dem entsprechenden Document-Registry-Aktor zuständig. Der Aktor speichert ausschließlich den binären Datenstrom der Dokumente; alle für die Registrierung an einem Document-Registry-Aktor notwendigen Metadaten müssen daher vom initiierenden Source-Aktor zugewiesen werden. Einzige Ausnahme hiervon bildet die URI, die der Repository-Aktor an jedes Dokument für spätere Abrufe durch Document-Consumer-Aktoren vergibt.

Patient Identity Source Der Patient-Identity-Source-Aktor stellt für jeden Patienten einen eindeutigen Identifikator zur Verfügung und ermöglicht es dem Document-Registry-Aktor Identifikatoren, die er bei Transaktionen mit anderen Knoten erhält, auf ihre Gültigkeit hin zu überprüfen.

2.1.4. Transaktionen

Die Aktoren interagieren gemäß [iti08b] mit Hilfe der folgenden Transaktionen.

Provide and Register Document Set Die Provide-and-Register-Document-Set-Transaktion wird durch den Document-Source-Aktor initiiert. Der empfangende Document-Repository-Aktor ist dafür verantwortlich, die als binären Datenstrom erhaltenen Dokumente persistent zu speichern und diese durch die ebenfalls erhaltenen Metadaten dem Document-Registry-Aktor mit Hilfe der Register-Document-Set-Transaktion bekannt zu machen. Abbildung 2.2 stellt diesen Ablauf in From eines Sequenzdiagramms dar.

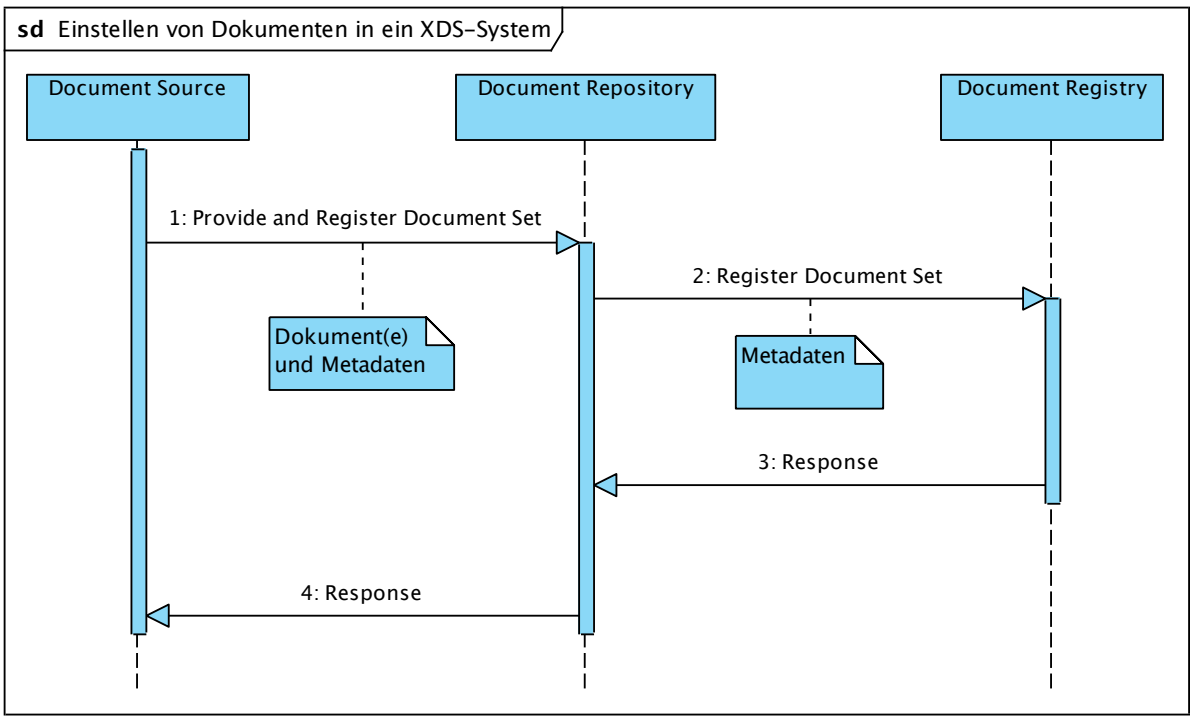

Abbildung 2.2.: Einstellen von Dokumenten in ein XDS-System

Register Document Set Die Register-Document-Set-Transaktion wird von einem Document-Repository-Aktor angestoßen, um Dokumente an einem Document-Registry-Aktor zu registrieren. Hierzu sendet der Repository-Aktor im Rahmen der Transaktion zu jedem Dokument, das registriert werden soll, die dazugehörigen Metadaten an den Registry-Aktor,

der diese auf ihre Gültigkeit und Standardkonformität hin überprüft, einen XDSDocumentEntry-Datensatz daraus erzeugt und diesen abspeichert. Sollte das Validieren der Metadaten bei einem oder mehreren Dokument fehlschlagen, so schlägt die komplette Transaktion fehl — und damit gegebenenfalls auch die auslösende Provide-and-Register-Document-Transaktion.

Query Registry Ein Dokument-Consumer-Aktor verwendet die Query-Registry-Transaktion, um einen Document-Registry-Aktor nach Dokumenten zu durchsuchen, die bestimmten Kriterien entsprechen. Sofern passende Dokumente gefunden werden, liefert der Registry-Aktor eine List der Metadaten dieser Dokumente zurück. Die Metadaten enthalten jeweils einen Verweis auf den Speicherort in einem Document-Repository-Aktor, mit dem das eigentliche Dokument unter Verwendung der Retrieve-Document-Transaktion lokal verfügbar gemacht werden kann.

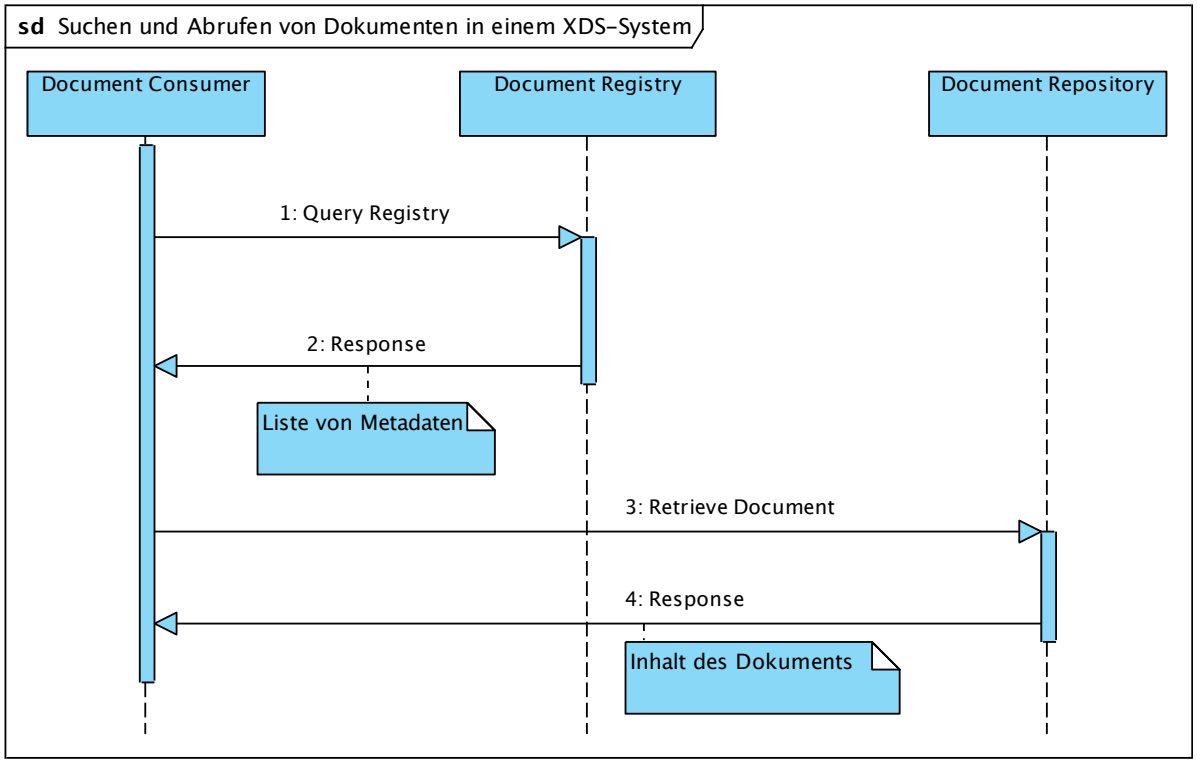

Abbildung 2.3.: Suchen und Abrufen von Dokumenten in einem XDS-System

Retrieve Document Die Retrieve-Document-Transaktion wird durch einen Document-Consumer-Aktor ausgelöst, um ein bestimmtes Dokument von einem Document-Repository-Aktor abzurufen. Der angesprochene Document-Repository-Aktor liefert als Antwort den Inhalt des gewünschten Dokuments zurück.

Patient Identity Feed Im XDS-Profil ist es Aufgabe der Patient-Identity-Feed-Transaktion den Document-Registry-Aktor mit in seiner Affinity-Domain gültigen Patienten-Identifikatoren zu beliefern.

2.2. Projekte und Institutionen im Umfeld von IHE-XDS

2.2.1. Open Healthcare Framework (OHF)

Das „Open Healthcare Framework“ (OHF)[1] ist ein Unterprojekt von Eclipse, welches Implementierungen für verschiedene medizinrelevante IT-Standards entwickelt und bereitstellt. Daran beteiligt sind neben IBM weitere Firmen und Institutionen aus dem Gesundheitswesen, so zum Beispiel die amerikanische Mayo Clinic.

Neben den Lösungen zu den IHE-Profilen ATNA, PDQ, PIX und XDS, bietet OHF zum Beispiel auch Implementierungen zu HL7v2 und v3.

Zum Zeitpunkt der Erstellung dieses Textes im Dezember 2008 befindet sich die Implementierungen der IHE-Profile im Übergang zu „Open Health Tools“[2], einer open-source Community, die sich der Förderung der Zusammenarbeit von Medizinern und IT-Kräften verschrieben hat. An der Verfügbarkeit der XDS-Implementierung ändert sich dadurch aber nichts.

2.2.2. National Institute of Standards and Technology (NIST)

Das 1901 gegründete National Institute of Standards and Technology (kurz NIST) ist eine staatliche, amerikanische Forschungseinrichtung mit Hauptsitz in Gaithersburg, Maryland ([NIS08]). Die Abteilung „Software and Systems“ beschäftigt sich im Rahmen eines eHealth-Programms unter anderem auch mit IHE-Profilen und entwickelt in diesem Zusammenhang einen open-source XDS-Server[3]. Eine Instanz dieses Servers wird von der NIST selbst als öffentlich im Internet zugänglicher Testserver betrieben, ist jedoch ebenso zum freien Download verfügbar und kann somit im lokalen Umfeld installiert werden. Das Softwarepaket umfasst einen Document-Registry- und Document-Repository-Aktor.

2.2.3. IBM Health Information Exchange (HIE)

Neben der Beteiligung an OHF betreibt IBM auch eine eigene XDS-Infrastruktur, genauer einen öffentlichen XDS-Server bestehend aus Registry- und Repository-Aktor. Dieser war jedoch zum Zeitpunkt der Erstellung dieses Textes Ende Dezember 2008 nicht mehr erreichbar.

[1] `http://eclipse.org/ohf`

[2] `http://www.openhealthtools.org/`. Die IHE-Profile finden sich zukünftig unter `https://iheprofiles.projects.openhealthtools.org/`.

[3] `http://ihexds.nist.gov/`

2.3. Weitere relevante IHE-Profile

2.3.1. Audit Trail and Node Authentication (ATNA)

Das Audit-Trail-and-Node-Authentication-Profil stellt einerseits Authentifizierungsmaßnahmen für andere IHE-Profile bereit und bietet außerdem durch den „Audit Trail"-Mechanismus eine Möglichkeit Sicherheitsereignisse, wie Systemzugriffe durch Benutzer, nachvollziehen zu können. Diese Funktionen sind zentral im ATNA-Profil zusammengefasst und werden deshalb in den einzelnen IHE-Profilen nicht neu definiert. So kann zum Beispiel ein XDS-Consumer-Aktor zur Authentifizierung gegenüber einem XDS-Registry-Aktor auf die von ATNA bereitgestellten Methoden zurückgreifen. Ebenso kann in einem XDS-System durch konsequente Nutzung des ATNA-Audit-Trails durch alle beteiligten Aktoren zum Beispiel festgestellt werden, welcher Benutzer über welchen Consumer-Aktor zu einer bestimmten Zeit ein bestimmtes Dokument abgerufen hat.

Es ist jedoch zu beachten, dass die IHE keine konkreten Maßnahmen oder Technologien zur lokalen Benutzerauthentifizierung an einem Knoten vorschreibt und das Augenmerk von ATNA nur auf der Authentifizierung von Knoten untereinander liegt. Abbildung 2.4 zeigt das Suchen und Abrufen von Dokumenten in einem XDS-System mit durch ATNA geschützten Knoten (auf die Darstellung der initialen Zeitsynchronisierung aller Knoten wurde der Übersichtlichkeithalber verzichtet). Hinweis: Ein sicherer bzw. geschützter Knoten umfasst im Sinne von ATNA nicht nur die konkrete Anwendungssicherheit, sondern bezieht explizit auch das Betriebssystem und die übrige zum Betrieb notwendige Softwareumgebung mit ein.

2.4. ebXML

2.4.1. Überblick

„Electronic Business using eXtensible Markup Language", oder kurz ebXML, bezeichnet eine Reihe von XML-basierten Spezifikationen zum standardisierten, elektronischen Datenaustausch im Rahmen von Geschäftsprozessen, die 1999 von OASIS und UN/-CEFACT auf den Weg gebracht wurden. Der XDS-Registry-Aktor implementiert eine Registry im Sinne der ebXML-Registry-Services. Dadurch liegt es nahe, dass auch das Datenmodell von XDS im Wesentlichen auf der ebXML-Registry-Information-Model-Spezifikation beruht. Als dritte Technologie aus der ebXML-Familie setzt XDS noch den „Message Service" ein. Diese Spezifikation ist jedoch — im Gegensatz zu den ersten beiden — für den weiteren Verlauf dieser Arbeit wenig interessant, weshalb hier nicht näher darauf eingegangen wird.

> HINWEIS: Sofern im Text nicht besonders erwähnt, bezeichnet *Registry* in diesem Kapitel eine Registry im Sinne von ebXML, nicht XDS.

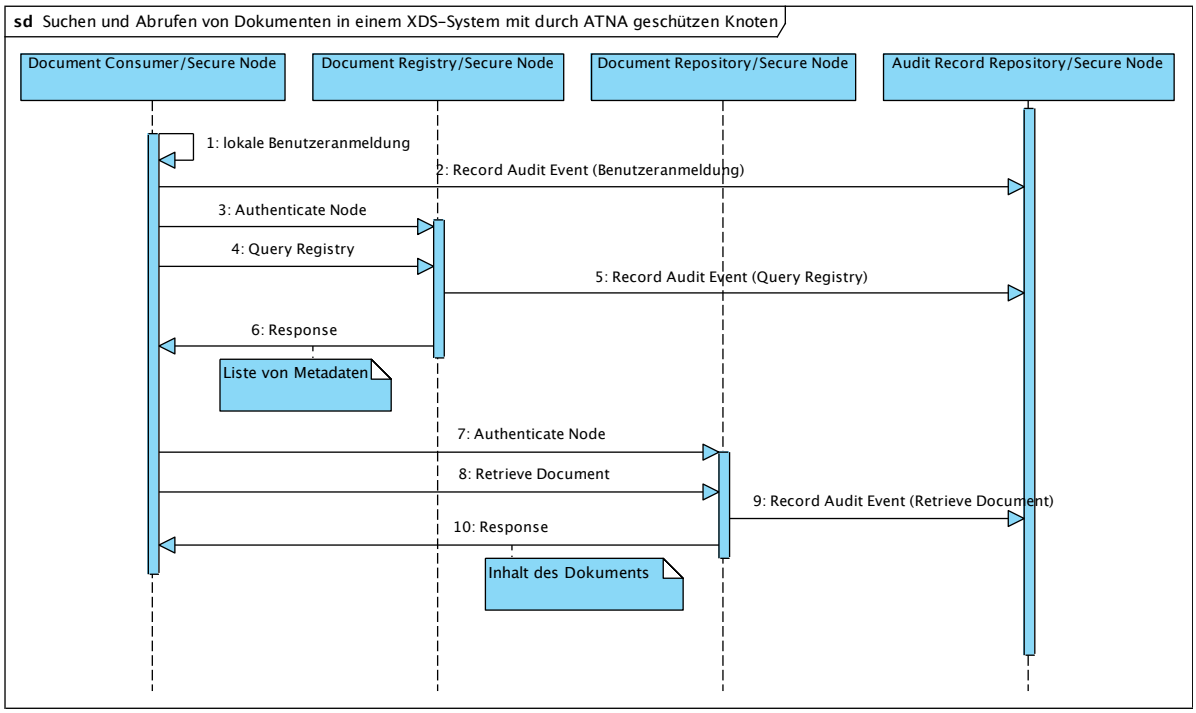

Abbildung 2.4.: Suchen und Abrufen von Dokumenten in einem XDS-System mit durch ATNA geschützten Knoten

[ebr01a] beschreibt eine Registry schlicht als persistenten Speicherort für Informationen. Zusätzlich existieren „Registry Services“, die eine Schnittstelle zum Zugriff auf die in einer Registry gespeicherten Informationen zur Verfügung stellen. Dazu bildet das „Registry Information Model“ ein Schema der Registry, in dem es definiert, welche Objekte in einer Registry gespeichert und wie diese dort organisiert sind.

2.4.2. ebRIM — Registry Information Model

Wie bereits eingangs erwähnt, bildet das „Registry Information Model“ ein Schema einer Registry. Das Information-Model kann einerseits verwendet werden, um das Schema einer von der Registry verwendeten Datenbank zu erzeugen. Zusätzlich können aus dem Information-Model auch Schnittstellen und Klassen für die Registry-Implementierung abgeleitet werden. In der Registry selbst findet das Information-Model also, wenn überhaupt, nur indirekt Verwendung, worauf [ebr01a] deutlich hinweist. In der Spezifikation wird weiter ausgeführt, dass das Information-Model keine bestimmte Implementierung vorschreibt, sondern eher zu beschreibenden Zwecken gedacht ist.

Die Registry-Information-Model-Spezifikation definiert als Teil seines Datenmodells eine Reihe von Klassen (Abbildung 2.5). Im Folgenden werden die für XDS relevanten Klassen näher vorgestellt:

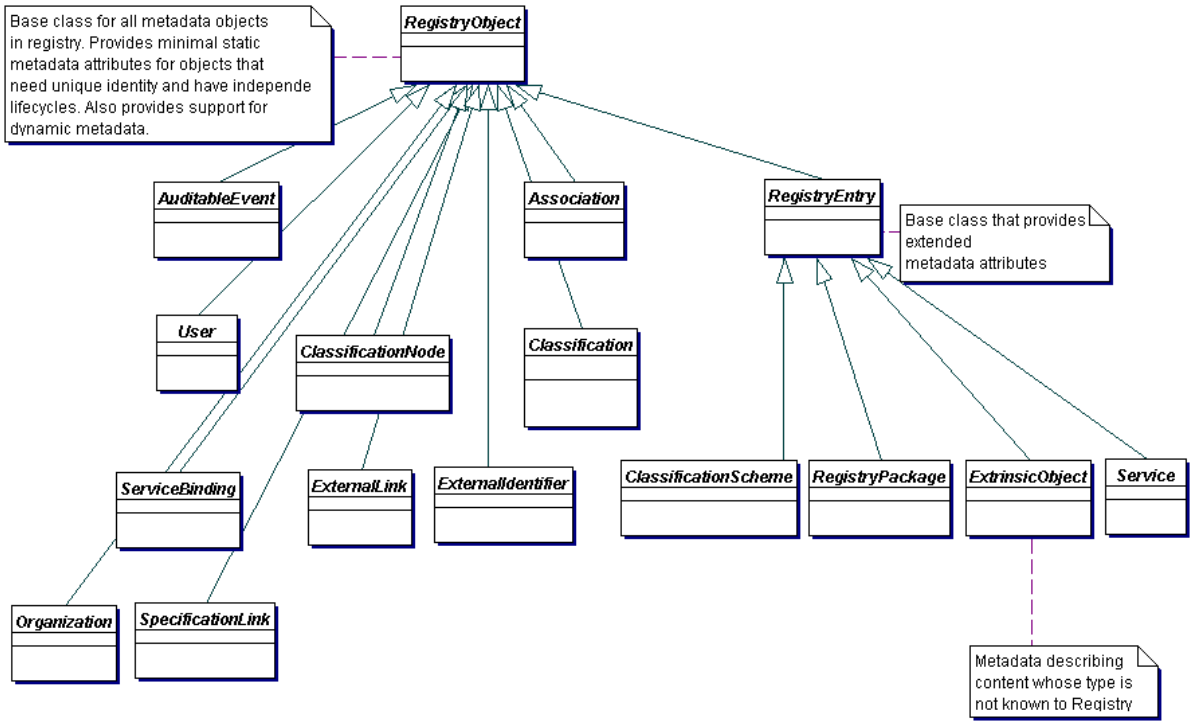

Abbildung 2.5.: Klassenhierarchie von ebRIM (aus [ebr01a])

RegistryObject `RegistryObject` ist eine abstrakte Basisklasse für Registry-Objekte, die einen grundlegenden Satz an Attributen definiert.

Slot Mit der Klasse `Slot` wird ein Mechanismus bereitgestellt, der es erlaubt, Objekte der Klasse `RegistryObject` dynamisch um beliebige Attribute zu erweitern, indem eine `RegistryObject`-Instanz beliebige `Slot`-Instanzen aggregieren kann.

RegistryPackage Instanzen dieser Klasse dienen der Gruppierung logisch zusammengehörender `RegistryObject`-Instanzen.

Classification, ClassificationScheme Eine Instanz der Klasse `ClassificationScheme` definiert die Struktur eines Klassifizierungsschemas, d.h. die Art, wie `RegistryObject`-Instanzen klassifiziert bzw. kategorisiert werden. Die Kategorie bzw. die Klasse, der eine Instanz vom Typ `RegistryObject` angehört, wird durch eine `Classification`-Instanz, die den Klassifizierungswert und einen Verweis auf das entsprechende Klassifizierungsschema enthält, festgelegt.

ExtrinsicObject `ExtrinsicObject`-Instanzen beschreibt Objekte, deren Struktur der Registry nicht bekannt ist, d.h. deren Aufbau nicht in der Registry selbst beschrieben ist, die aber trotzdem in der Registry abgelegt werden sollen.

Association Instanzen vom Typ `Association` stellen N:M-Verknüpfungen zwischen `RegistryObject`-Instanzen her. Der Typ der Verknüpfung wird über ein entsprechendes Attribut festgelegt.

Internationalisierung von Zeichenfolgen Zur Internationalisierung von Zeichenfolgen (Strings) bringt ebRIM zwei Klassen mit: `InternationalString` und `LocalizedString`. Ein `LocalizedString`-Objekt bringt die jeweils lokalisierte Zeichenfolge mit einer Sprachbezeichnung in Verbindung. Ein `InternationalString`-Objekt aggregiert dann alle Lokalisierungen, also `LocalizedString`-Objekte, einer Zeichenfolge und stellt eine Schnittstelle zum Zugriff auf den String einer bestimmten Sprache zur Verfügung.

2.4.3. ebRS — Registry Services

Während ebRIM das Schema einer Registry beschreibt, definiert ebRS, die „ebXML Registry Services“, die Schnittstellen und Methoden, die eine Registry zum Zugriff und Management ihres Inhaltes anbieten muss. Nach [ebr01b] muss eine ebXML Registry mindestens zwei Schnittstellen abieten: Die sogenannte *Life-Cycle-Management-* und die *Query-Management*-Schnittstelle. Erstere stellt Methoden bereit, die es beispielsweise erlauben, Objekte in der Registry anzulegen, vorhandene Objekte zu verändern, zu löschen oder deren Status zu ändern; letztere dient dem Suchen und Abrufen von in der Registry gespeicherten Informationen. Beide Schnittstellen müssen entweder über SOAP, den „ebXML Message Service“ oder beide Techniken ansprechbar sein. Eine dritte Schnittstelle ist für den Registry-Client vorgesehen. Diese wird in jeder Anfrage übergeben und dient der Verarbeitung der Registry-Antwort.

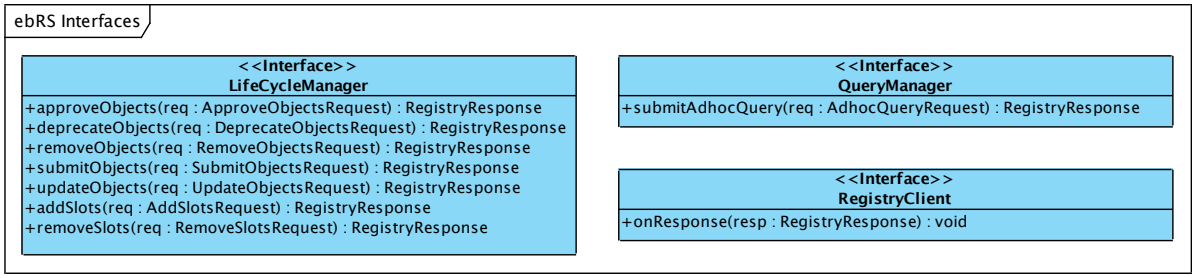

Abbildung 2.6.: Schnittstellen der ebXML Registry Services

Abbildung 2.6 zeigt die erwähnten Schnittstellen inklusive der zur Verfügung gestellten Methoden. Die Query-Management-Schnittstelle erlaubt das Suchen über eine sogenannte XML-basierte *FilterQuery* bzw. alternativ die Formulierung von Anfragen in

einer SQL-Syntax; letzteres ist allerdings optional und muss nicht zwingend von jeder Registry-Implementierung unterstützt werden. Listing 2.1 zeigt eine Beispiel SQL-Anfrage, die die Menge der `ExtrinsicObject`-Instanzen zurückliefert, deren Name „Acme“ enthält und deren Version größer als 1.3 ist.

```
1 SELECT eo.id from ExtrinsicObject eo, Name nm WHERE
      nm.value LIKE '%Acme%' AND
3     eo.id = nm.parent AND
      eo.majorVersion >= 1 AND
5     (eo.majorVersion >= 2 OR  eo.minorVersion > 3);
```

Listing 2.1: Beispiel einer ebRS-SQL-Anfrage (aus [ebr01b])

Zusätzlich ist es möglich den Typ der Rückgabewerte einer Anfrage festzulegen. So kann beispielsweise bestimmt werden, dass nur Objektreferenzen oder nur Unterobjekte von RegistryEntry zurückgeliefert werden.

3. Anforderungsanalyse

3.1. Anforderungen an ein System gemäß IHE-XDS

Ein XDS-System gemäß [iti08a, iti08b] benötigt genau einen Document-Registry-Aktor, mindestens einen Document-Repository-Aktor sowie ein oder mehrere Document-Source- und Document-Consumer-Aktoren. Die Spezifikation erlaubt auch die Zusammenfassung des Source- und Repository-Aktors zu einer logischen Instanz, die beide Schnittstellen unterstützt. Jeder Aktor muss die für ihn relevanten, im vorherigen Kapitel beschriebenen Transaktionen unterstützen, die Anfrage- und Antwortdaten dafür gemäß der Beschreibung in [iti08b] nach ebXML codieren bzw. aus ebXML decodieren sowie diese Anfragen über SOAP oder SMTP versenden können.

Der Repository-Aktor muss durch geeignete Maßnahmen sicherstellen, dass die durch Provide-and-Register-Document-Set-Transaktionen übermittelten Dokumente persistent gespeichert werden und jederzeit über den durch ihn vergebenen URI abrufbar sind.

Der Registry-Aktor muss die an ihn im Rahmen von Register-Document-Set-Transaktionen übergebenen Metadaten auf ihre Gültigkeit hin überprüfen. Sind die Daten gültig, muss der Aktor dafür Sorge tragen, dass die Metadaten persistent gespeichert werden und jederzeit durch Query-Registry-Transaktionen auffindbar und abrufbar sind. Die Beschreibung der Transaktionen in [iti08b] impliziert, dass der Registry-Aktor eine Registry im Sinne von ebXML implementieren muss. Dies bedeutet, dass ebRS-konforme Schnittstellen zur Verfügung stehen müssen und der Aktor ebRIM-Objekte verarbeiten kann.

Die Registry-Document-Set-Transaktion entspricht im wesentlichen der „SubmitObjects“-Methode der Life-Cycle-Manager-Schnittstelle einer ebXML-Registry, mit der Ausnahme, dass neben der Konformität zu ebXML auch die XDS-spezifischen Anforderungen an die Gültigkeit der Daten (z.B. benötigte Attribute, flache Hierarchie von XDSFolder-Objekten, usw.) überprüft werden müssen. Als Anfrageprotokoll wird SOAP verwendet, welches ein „SubmitObjectsRequest“-Element enthält, das wiederum die zur Speicherung bestimmten Metadaten im ebRIM-Format enthält.

Die Provide-and-Register-Document-Set-Transaktion verläuft weitgehend analog. Da hier jedoch gewöhnlich mehrere Elemente (SOAP-codierte Anfrage/Metadaten und ein oder mehrere Dateien) in einer Anfrage übertragen werden müssen, kommt „SOAP with Attachments“ zum Einsatz, welches die einzelnen Elemente mit Hilfe von Multipart-MIME codiert, um sie dann in einer einzigen HTTP-Anfrage zu übertragen. Siehe Listing 3.1 für eine entsprechende Beispielanfrage.

```
POST /ebxmlrr/registry/soap HTTP/1.1
Content-Type: multipart/related; type="text/xml"; boundary=--
    boundary01
SOAPAction: ""
[...]

--boundary01
Content-Type: text/xml

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
    soap/envelope/">
<SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <SubmitObjectsRequest xmlns="urn:oasis:names:tc:ebxml-
        regrep:registry:xsd:2.1">
      <LeafRegistryObjectList>
        <ExtrinsicObject id="doc_1" mimeType="text/xml"/>
      </LeafRegistryObjectList>
    </SubmitObjectsRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--boundary01
Content-Type: text/xml
Content-Id: <doc_1>

<books>
  <book isbn="0345374827"><title>The Great Shark Hunt</title>
  <author>Hunter S. Thompson</author></book>
  <book><title>Life with Father</title><author>Clarence Day</
      author></book>
</books>

--boundary01--
```

Listing 3.1: Beispiel einer Provide-and-Register-Document-Set-Anfrage (aus [xds07c])

Für die Query-Registry-Transaktion kommt dagegen wieder einfaches SOAP zum Einsatz. Diese Transaktion entspricht der „submitAdhocQuery“-Methode der Query-Manager-Schnittstelle einer ebXML-Registry. XDS verwendet jedoch ausschließlich Anfragen in der nach [ebr01b] optionalen SQL-Syntax. Hier wird ein „AdhocQueryRequest“-Element, welches die Suchanfrage enthält, per SOAP codiert und über eine HTTP-Anfrage versendet. Folgender Code (Listing 3.2) zeigt eine mögliche Suchanfrage.

```
<AdhocQueryRequest>
    <ResponseOption returnType = "LeafClass"
        returnComposedObjects="true"/>
    <SQLQuery>
    SELECT eo.id, [...] FROM ExtrinsicObject eo,
        ExternalIdentifier ei
         WHERE
              eo.id = ei.registryobject AND
              ei.identificationScheme='urn:uuid:58a6f841-87b3-4
                  a3e-92fd-a8ffeff98427' AND
              ei.value='NIST-5678^^^&1.3.6.1.4.1.21367.2005.1.1&
                  ISO^PI' AND
              eo.status = 'Approved'
    </SQLQuery>
</AdhocQueryRequest>
```

Listing 3.2: Beispiel einer Query-Registry-Anfrage (ohne SOAP-Codierung; aus [xds07a])

Eine Endanwendung soll jedoch explizit keine SQL-Anfragen formulieren müssen ([iti08b], Kapitel 3.16.4.1.4). Vielmehr ist es Aufgabe des Consumer-Aktors einen Satz von Suchanfragen bereitzustellen. Hierfür definiert die Spezifikation einen minimalen Anfragekatalog, auf dessen Unterstützung durch den Registry-Aktor sich der Consumer-Aktor verlassen können muss. Dies bedeutet, dass der Registry-Aktor die SQL-Syntax in dem Umfang implementieren muss, dass alle im Anfragekatalog spezifizierten Suchanfragen ausgeführt werden können. Tabelle 3.1 beschreibt alle Suchanfragen, die im minimalen Anfragekatalog enthalten sind.

Wird eine Standard-ebXML-Registry-Implementierung verwendet, so muss diese zuvor geeignet initialisiert werden, d.h. es müssen die XDS-spezifischen Typen und Attribute bekannt gemacht werden. Siehe Tabelle 3.2 für einen Ausschnitt der Liste der anzulegenden Objekte. Diese Daten entstammen den Tabellen aus [xds07b]; dort sind auch die übrigen Werte zu finden.

3.2. Teststand-Anforderungen

Um die in einem XDS-Repository gespeicherten Dokumente zu warten, wird ein Client benötigt (nachfolgend *GUI-Client* genannt), über dessen grafische Oberfläche einfache Anfragen an eine XDS-Registry gestellt, gefundene Dokumente aus einem XDS-Repository abgerufen sowie neue Dokumente in ein XDS-Repository eingestellt werden können. Dieser Client soll auch genutzt werden, um andere XDS-Implementierungen auf ihre Kompatibilität hin zu evaluieren. Zur Evaluation von XDS-Source- und XDS-

Consumer-Implementierungen wird ein Serversystem benötigt, dass die Rollen einer XDS-Registry und eines XDS-Repositories einnimmt.

3.2.1. Anforderungen an den GUI-Client

Funktionale Anforderungen Folgende funktionale Anforderungen müssen durch den GUI-Client mindestens erfüllt werden:

(R1) Über die grafische Oberfläche soll eine Suchanfrage an eine XDS-Registry gestellt werden können, bei der alle der Registry bekannten Dokumente (DocumentEntry-Objekte) zu einer durch den Benutzer angegebenen Patienten-ID zurückgeliefert werden.

(R2) Zu jedem durch eine Suchanfrage zurückgelieferten Dokument soll es möglich sein, die Metadaten Titel (in der ersten verfügbaren Sprache), Class-, Confidentiality-, Format-, Healthcare-Facility-, Practice-Setting- und Type-Code anzeigen zu lassen.

(R3) Es soll eine Funktion bereitgestellt werden, durch die ein durch eine Suchanfrage zurückgeliefertes Dokument von dem entsprechenden Repository auf das lokale System des Benutzers übertragen und dort im Dateisystem gespeichert wird.

(R4) Über die grafische Oberfläche soll ein lokales Dokument in ein XDS-Repository übertragen werden können und seine vom Benutzer angegebenen Metadaten in der zugehörigen Registry abgelegt werden. Der Benutzer soll folgende Daten über die grafische Oberfläche bestimmen können: Patienten-ID, Titel des Submission-Set, Content-Type des Submission-Set, Titel des Dokuments, Class-, Confidentiality-, Format-, Healthcare-Facility-, Practice-Setting- und Type-Code des Dokuments.

(R5) Muss der Benutzer einen Dateipfad angeben, so soll dies ausschließlich über einen entsprechenden Auswahldialog erfolgen.

(R6) Wo nur die Angabe von einer bestimmten Menge an Werten durch den Benutzer erlaubt ist, soll dies durch eine entsprechend vorbelegte Auswahlliste realisiert werden.

(R7) Es soll die Möglichkeit bestehen, Adressen von XDS-Registry- und XDS-Repository-Aktoren im laufenden Programm einzugeben bzw. zu ändern.

(R8) Auch soll es möglich sein, Patienten-IDs im laufenden Programm einzugeben.

Nicht-funktionale Anforderungen Darüber hinaus muss der GUI-Client mindestens folgende nicht-funktionale Anforderungen erfüllen:

(R9) Es ist ausreichend, wenn nur eine Aktion (Einstellen von Dokumenten oder Suchen von Dokumenten) gleichzeitig durchgeführt werden kann; jedoch soll die Oberfläche während des Abarbeitens einer Aktion weiter auf Benutzerinteraktion reagieren.

(R10) Es soll vorerst genau ein Dokument pro Vorgang in ein Repository eingestellt werden können.

(R11) Der Client muss auch über eine durch SSL verschlüsselte Verbindung mit einem Serversystem Kontakt aufnehmen können.

(R12) Der Client agiert in seiner Rolle als XDS-Source gegenüber einem XDS-Repository als genau ein bestimmter, voreingestellter Benutzer. Der Client muss durch die Oberfläche jedoch keine unterschiedlichen Benutzerkonten unterstützen.

Anfrage	Beschreibung
FindDocuments	Sucht nach XDSDocumentEntry-Objekte mit einer bestimmten Patienten-Id und weiteren optionalen Parametern.
FindSubmissionSets	Sucht nach XDSSubmissionSet-Objekte mit einer bestimmten Patienten-Id und weiteren optionalen Parametern.
FindFolders	Sucht nach XDSFolder-Objekte mit einer bestimmten Patienten-Id und weiteren optionalen Parametern.
GetAll	Liefert alle Objekte zu einem bestimmten Patienten.
GetDocument	Liefert ein bestimmtes XDSDocumentEntry-Objekt.
GetSubmissionSetAndContents	Liefert ein bestimmtes XDSSubmissionSet-Objekt inklusive dessen Inhalt.
GetFolderAndContents	Liefert ein bestimmtes XDSFolder-Objekt inklusive dessen Inhalt.
GetFoldersFromDocument	Liefert alle XDSFolder-Objekte, die das angegebene XDSDocumentEntry-Objekt beinhalten.
GetRelatedDocuments	Liefert alle XDSDocumentEntry-Objekte, die zu dem angegebenen XDSDocumentEntry-Objekt über ein Association-Objekt in Beziehung stehen.
GetFolders	Liefert bestimmte XDSFolder-Objekte.
GetAssociations	Liefert Association-Objekte, die mit dem angegebenen XDSDocumentEntry-Objekt in Beziehung stehen.
GetDocumentsAndAssociations	Liefert bestimmte XDSDocumentEntry-Objekte und die Assoication-Objekte, die mit diesen in Beziehung stehen.
GetSubmissionSets	Liefert die XDSSubmissionSet-Objekte, mit denen die angegebenen XDSDocumentEntry- oder XDSFolder-Objekte übertragen wurden.

Tabelle 3.1.: Minimaler Anfragekatalog, für eine genauere technische Beschreibung der Anfragen siehe [iti08b], Kapitel 3.16.4.1.4

UUID	Name	Typ
urn:uuid:a54d6aa5-d40d-43f9-88c5-b4633d873bdd	XDSSubmissionSet	ClassificationNode
urn:uuid:6b5aea1a-874d-4603-a4bc-96a0a7b38446	XDSSubmissionSet.patientId	External Identifier
urn:uuid:7edca82f-054d-47f2-a032-9b2a5b5186c1	XDSDocumentEntry	ClassificationNode
urn:uuid:41a5887f-8865-4c09-adf7-e362475b143a	XDSDocumentEntry.classCode	ExternalClassification-Scheme
urn:uuid:58a6f841-87b3-4a3e-92fd-a8ffeff98427	XDSDocumentEntry.patientId	ExternalIdentifier
urn:uuid:d9d542f3-6cc4-48b6-8870-ea235fbc94c2	XDSFolder	ClassificationNode
urn:uuid:917dc511-f7da-4417-8664-de25b34d3def	APND	ClassificationNode

Tabelle 3.2.: Auswahl von UUIDs für XDS-Datentypen und Attribute

3.2.2. Anforderungen an das Serversystem

Funktionale Anforderungen Das Serversystem des Teststands muss mindestens die folgenden funktionalen Anforderungen erfüllen:

(R13) Das Serversystem muss die Rolle einer XDS-Registry gemäß IHE-XDS-Standard und im obigen Abschnitt beschriebener Anforderungen unterstützen, d.h. die benötigten Schnittstellen und Funktionen zur Verfügung stellen.

(R14) Das Serversystem muss die Rolle eines XDS-Repositories gemäß IHE-XDS-Standard und im obigen Abschnitt beschriebener Anforderungen unterstützen.

4. Stand der Technik

4.1. Überblick

Dieses Kapitel beschreibt freie oder unter Open-Source-Lizenz verfügbare Programme oder Komponenten, die eine oder mehrere der im vorherigen Kapitel beschriebenen Anforderungen an ein XDS-System erfüllen.

4.2. Open Healthcare Framework

Der Open-Healthcare-Framework (OHF) bietet unter anderem auch eine Implementierung des XDS-Profils. Diese Implementierung enthält jedoch nur die „client-seitigen" Document-Source- und Document-Consumer-Aktoren, nicht jedoch einen Document-Registry- und Document-Repository-Aktor. Die Implementierungen der IHE-Profile erscheinen unter der „Eclipse Public License" (EPL) und liegen somit auch im Quelltext vor.

Die Client-Aktoren sind in Java implementiert und unterstützen, wie gefordert, die Query-Registry- und Retrieve-Document- (Consumer-Aktor) sowie die Provide-and-Register-Document-Set-Transaktionen (Source-Aktor). Darüber hinaus bietet das Projekt auch eine Abbildung des XDS-Metadatenmodells, wie es von einem Registry-Aktor verwendet wird, auf Java-Klassen und Schnittstellen mit Hilfe des „Eclipse Modeling Frameworks" (EMF). Ebenfalls erhältlich ist eine ATNA-Implementierung, die auch von den OHF-XDS-Aktoren zum Auditing verwendet wird. Die folgenden Abschnitte geben einen detaillierten Überblick über die einzelnen OHF-XDS-Komponenten.

Abbildung der Metadaten Durch OHF enthält der Entwickler neben den reinen Aktor-Implementierungen auch eine komplette Abbildung der XDS-Metadaten-Struktur auf Java-Schnittstellen und Klassen. Wie bereits erwähnt, erfolgt diese Abbildung mit Hilfe des „Eclipse Modeling Frameworks", weshalb an einigen Stellen auch dessen Klassen bzw. Schnittstellen auftauchen (z.b. `EList`).

So biete OHF neben den XDS-Datentypen *XDSSubmissionSet* (`SubmissionSetType` in OHF), *XDSDocumentEntry* (`DocumentEntryType`) und *XDSFolder* (`FolderType`) in Abbildung 4.1 auch weitere Datentypen aus dem HL7- bzw. ebXML-Standard, beispielsweise `CX` zur Codierung von Patienten-IDs, `InternationalStringType` und `LocalizedStringType` zur Lokalisierung von Zeichenfolgen und `CodedMetadataType` als Abbildung des ebXML-Codierungs-Schemas (siehe 4.2 für die Schnittstellen einiger exemplarischer Typen).

Aus dem praktischen Einsatz ergeben sich einige Anmerkungen zur Verwendung der genannten Typen. So ist bei `CX` darauf zu achten, dass für jede Zuweisung zu unterschiedlichen `DocumentEntryType`- und `SubmissionSetType`-Objekten ein neues `CX`-Objekt verwendet wird (d.h. es muss zumindest explizit kopiert/geklont werden). Ein erneutes Zuweisen eines bereits an anderer Stelle referenzierten `CX`-Objekts führt nicht zum Kopieren der Daten, sondern zur Verschiebung an die neue Stelle (dies ergibt sich aus der späteren Serialisierung zu XML). Das Beispiel in Listing 4.1 verdeutlicht das Problem. Würde man stattdessen in Zeile 18 `subset.setPatientId(docPatientId);` schreiben, hätte dies zur Folge, dass in der sich ergebenden ebRS-Anfrage die Patienten-ID nur im Submission-Set hinterlegt wäre, jedoch nicht bei den Dokument-Metadaten.

```
1  // txnData ist ein SubmitTransactionData-Objekt

3  DocumentEntryType docEntry = txnData.getDocumentEntry(
       docEntryUUID);

5  CX docPatientId = Hl7v2Factory.eINSTANCE.createCX();

7  docPatientId.setIdNumber(...);
   ...
9
   docEntry.setPatientId(docPatientId);
11
   CX submissionSetPatientId = Hl7v2Factory.eINSTANCE.createCX();
13 submissionSetPatientId.setIdNumber(...);
   ...
15
   SubmissionSetType subset = txnData.getSubmissionSet();
17 // falsch wäre hier subset.setPatientId(docPatientId);!
   subset.setPatientId(submissionSetPatientId);
```

Listing 4.1: Korrekte Verwendung des `CX`-Objekts

Die nicht explizit dokumentierte korrekte Verwendung von `InternationalStringType`- und `LocalizedStringType`-Objekten zeigt Listing 4.2.

```
   InternationalStringType is = MetadataFactory.eINSTANCE.
       createInternationalStringType();
2  LocalizedStringType ls = MetadataFactory.eINSTANCE.
       createLocalizedStringType();
   ls.setCharset("utf-8");
4  ls.setLang("en");
```

```
ls.setValue("Hello World!");
is.getLocalizedString().add(ls);
```

Listing 4.2: Korrekte Verwendung von `InternationalStringType`- und `LocalizedStringType`-Objekten

Wie aus den Abbildung ersichtlich, setzt OHF konsequent auf die Trennung von Schnittstelle und konkreter Implementierung, weshalb natürlich auch ein Factory-Muster (vgl. [GHJV95], S. 107) zur Erstellung der Daten-Objekte zum Einsatz kommt (siehe Abbildung 4.3).

Consumer-Aktor Der OHF-Consumer-Aktor bildet die Funktionalität eines XDS-Aktors in Java ab. Hierbei wurden die unterstützten Transaktionen „Query Registry“ und „Retrieve Document“ als Methoden der Consumer-Schnittstelle realisiert (siehe Abbildung 4.4). Die `Query`-Methode bietet entweder die Möglichkeit direkt über einen SQL-Ausdruck zu suchen oder, wie vom Standard eigentlich vorgesehen, über vordefinierte Query-Objekte, die Suchanfragen aus dem minimalen Anfragekatalog realisieren. OHF bringt hier momentan nur die Unterstützung für zwei Anfragen mit, nämlich `GetDocuments` und `FindDocuments`. Die übrigen Anfragen aus dem minimalen Anfragekatalog werden zum jetzigen Zeitpunkt nicht direkt unterstützt (sie ließen sich jedoch direkt über die äquivalente SQL-Anfrage durchführen). Der Parameter `returnReferencesOnly` der `Query`-Methode bestimmt, ob nur Objektreferenzen oder komplette Objekte zurückgeliefert werden.

Aus dem Rückgabewert kann der Erfolg oder gegebenenfalls der Fehlerzustand der Anfrage ersehen werden und es können im Erfolgsfall die gefunden Objekte abgefragt werden.

Source-Aktor Der OHF-Source-Aktor bildet die Provide-and-Register-Document-Set-Transaktion ebenfalls als eine Methode seiner Schnittstelle ab (siehe Abbildung 4.5). Die `Submit`-Methode akzeptiert einen Parameter von Typ `SubmitTransactionData`, der die Daten (Metadaten und Dateien) einer Transaktion kapselt. Zu diesem lassen sich Dokumente und Verzeichnisse hinzufügen sowie die Attribute des Submission-Set-Objekts bearbeiten. Über den Rückgabewert können der Erfolg der Anfrage bzw. eventuell aufgetretene Fehler bestimmt werden.

Es ist zu beachten, dass OHF selbst keine Überprüfung der Gültigkeit der angegebenen Daten durchführt, diese erfolgt erst durch den Registry-Aktor. Es ist also darauf zu achten, alle laut XDS-Spezifikation nötigen Attribute mit gültigen Werten zu füllen.

Bei beiden Aktoren erfolgt die Adressierung des Registry- bzw. Repository-Aktors durch Angabe der URI des SOAP-Webservice-Endpunkts.

Abbildung 4.1.: Abbildung der XDS-Datentypen in OHF

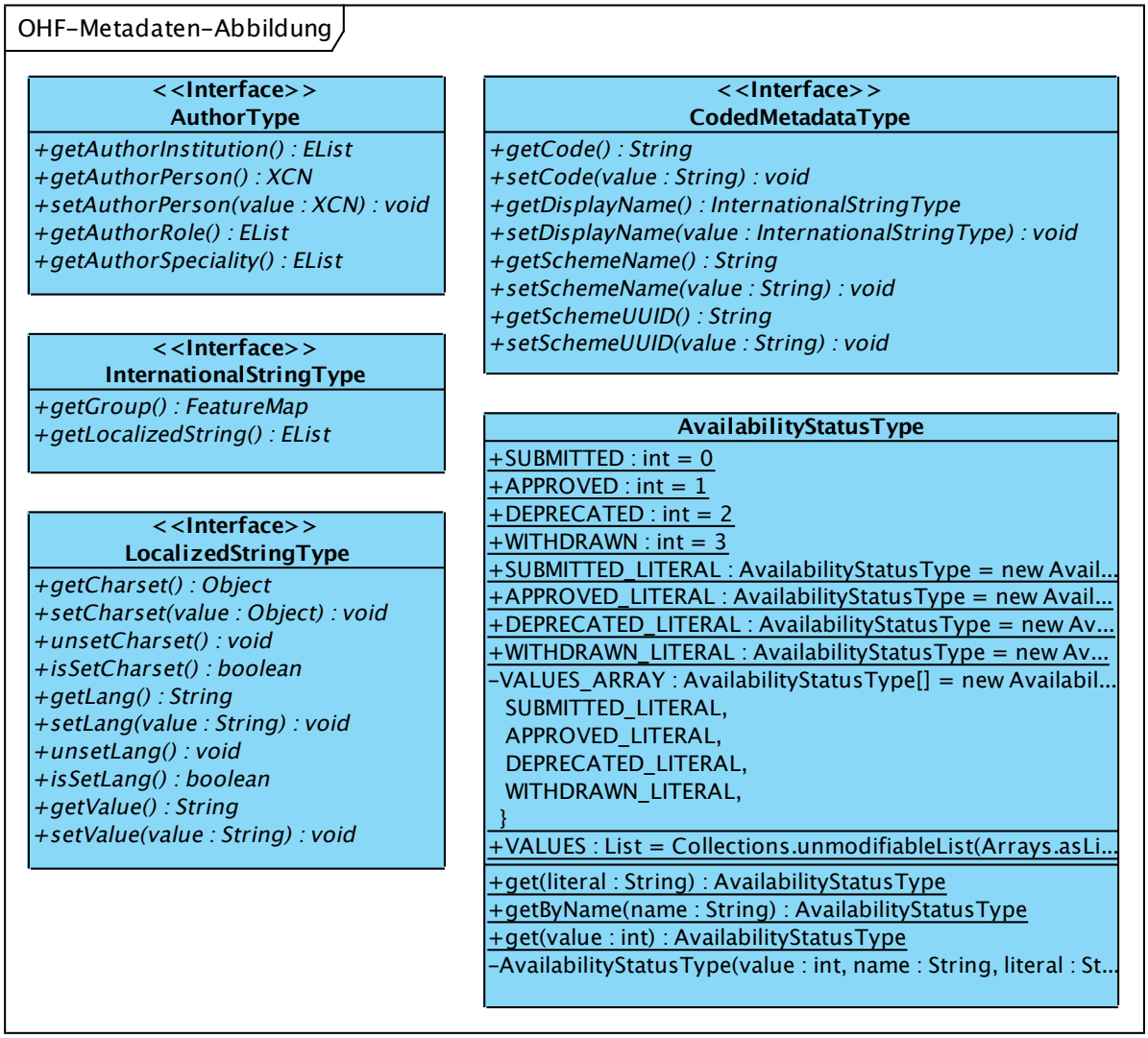

Abbildung 4.2.: Abbildung weiterer Metadatentypen in OHF

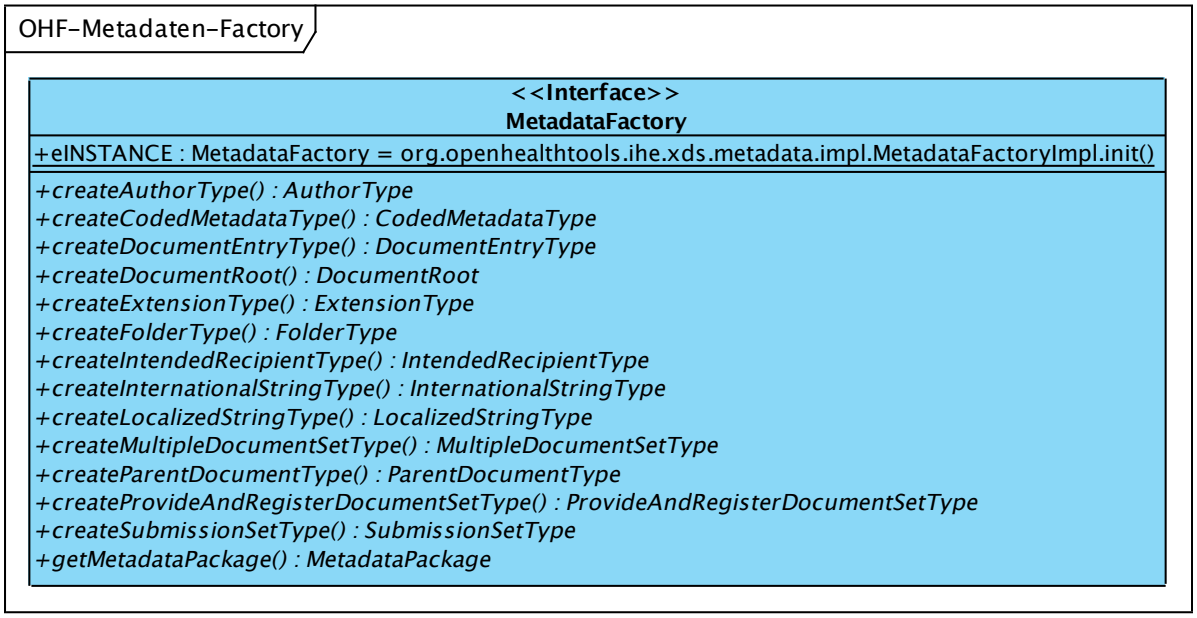

Abbildung 4.3.: OHF-Metadaten-Factory zur Erzeugung von Metadaten-Objekten

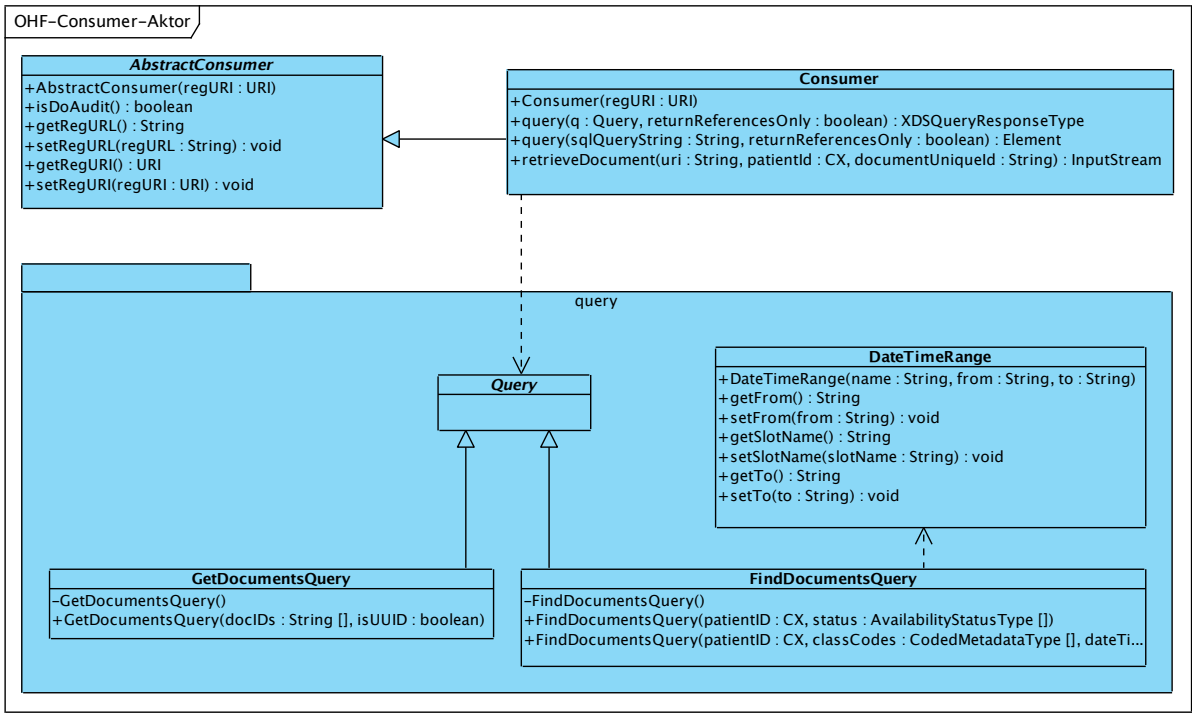

Abbildung 4.4.: OHF-Consumer-Aktor und abhängige Klassen/Schnittstellen

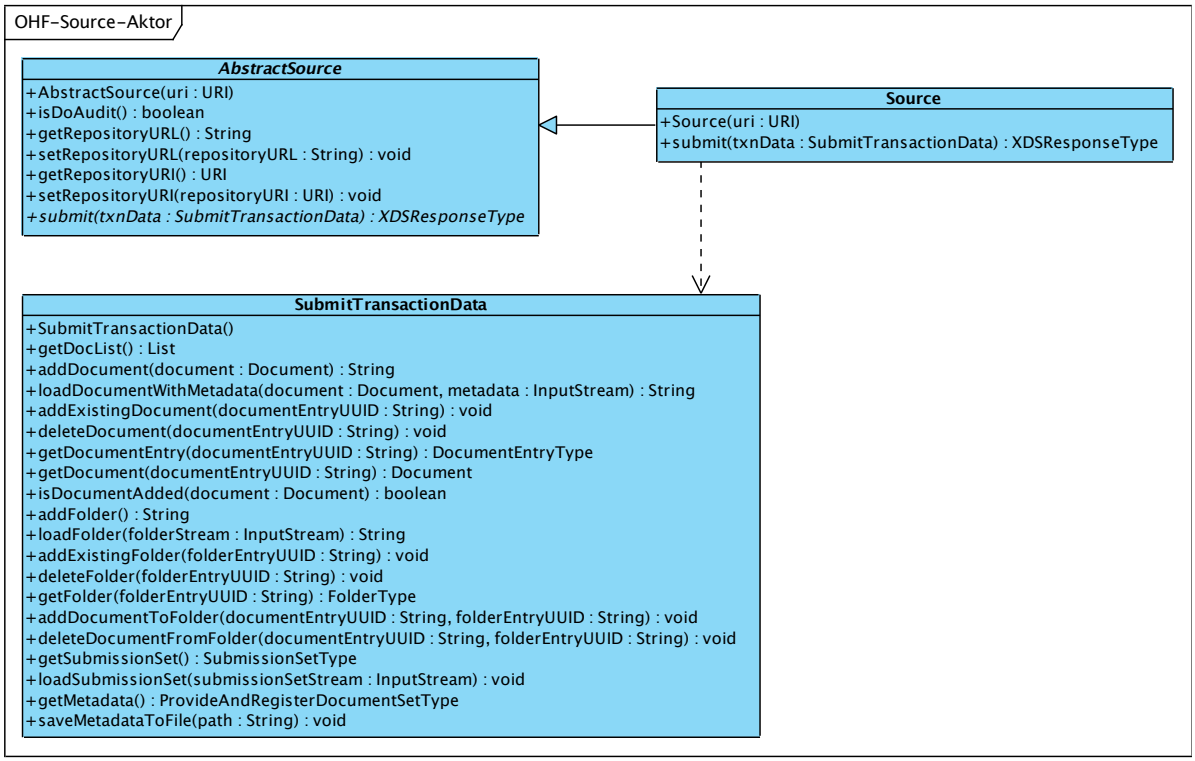

Abbildung 4.5.: OHF-Source-Aktor und abhängige Klassen/Schnittstellen

5. Grobarchitektur

5.1. Gesamtsystem

Gemäß der Anforderungsdefinition soll ein Gesamtsystem entstehen, welches die vier XDS-Aktoren beinhaltet. Hierzu soll ein GUI-Client entwickelt werden, welcher die Funktion eines Source- und Consumer-Aktors mit Hilfe der OHF-XDS-Implementierung realisiert. Zusätzlich wird eine Serverkomponente benötigt, welche einen Registry- und einen Repository-Aktor zur Verfügung stellt. Abbildung 5.1 gibt einen Überblick über die Struktur des Gesamtsystems. Für die Serverkomponente stehen mehrere Entwurfsalternativen zur Verfügung, die in den folgenden Abschnitten diskutiert werden.

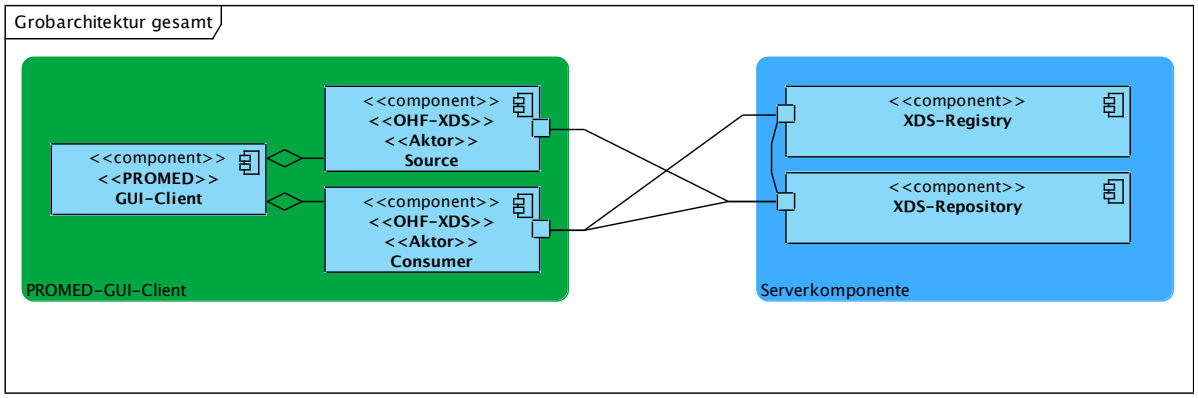

Abbildung 5.1.: Grobarchitektur des Gesamtsystems

5.2. Entwurfsalternativen für Serversubsystem

Das Serversubsystem soll die Funktion eines Registry- und Repository-Aktors übernehmen. Die Aufgaben hierfür lassen sich im wesentlichen auf mehrere Schichten verteilen.

1) Zuerst wird eine Schicht benötigt, die die für XDS notwendigen SOAP-Schnittstellen nach außen hin anbietet.

2) Danach muss eine Schicht folgen, die die XDS-spezifische Funktionalität, wie das Überprüfen der Metadaten auf ihre Gültigkeit und das Abspeichern und Auffinden

der Dateien (als Repository-Aktor) bzw. der Metadaten (als Registry-Aktor) an die dritte Schicht delegiert. Dies kann entweder direkt durch Weitergabe der ebXML-codierten Anfragen geschehen oder indirekt, indem die zweite Schicht die ebXML-codierten Anfragen interpretiert und vor der Weitergabe geeignet aufbereitet. Diese Schicht realisiert sozusagen die „Business Logic“ der Anwendung.

3) Die dritte Schicht sollte eine Abstraktion des persistenten Speichersystems sein und das persistente Abspeichern und Auffinden der Daten erleichtern. Idealerweise geht die Schicht auf die strukturellen Anforderungen von XDS ein. Im folgenden wird diese Schicht als Zugriffsschicht bezeichnet.

4) Zu guter letzt wird natürlich noch ein persistenter Datenspeicher benötigt, der die Daten, die durch die dritte Schicht verarbeitet werden, jederzeit zuverlässig wieder zur Verfügung stellen kann. Dabei kann es sich zum Beispiel um eine Datenbank oder das Dateisystem handeln. Hier können auch mehrere verschiedene Systeme zum Einsatz kommen, so wäre es zum Beispiel eventuell sinnvoll die Metadaten in einer Datenbank zu speichern, während die eigentlichen Dateien, die der Repository-Aktor verwaltet, im Dateisystem abgelegt werde.

Allgemein gehen die folgenden Abschnitte hauptsächlich auf die Realisierung des Registry-Aktors ein, da dieser die eigentliche „Herausforderung“ darstellt. Der Repository-Aktor ist technisch dagegen relativ trivial zu implementieren, beschränkt sich seine Aufgabe doch weitgehend darauf, die multipart-MIME-codierten Dateien aus einer Anfrage herauszulösen und presistent zu speichern bzw. diese Dateien auf „Zuruf“ wieder herauszugeben, was kein tieferes Wissen über die Struktur der ebXML-Anfragen erfordert. Die wenigen nötigen Eingriffe, beispielsweise das Hinzufügen des Hash-Wertes oder der URI zu den Metadaten lassen sich relativ leicht durch einfache XML-Operationen umsetzen.

5.2.1. Realisierung mit JackRabbit

Jackrabbit ist ein sogenanntes „Content Repository“, d.h. ein Datenspeicher, der sowohl strukturierte als auch unstrukturierte Inhalte in einer Hierarchie verwaltet. Zusätzlich bietet Jackrabbit beispielsweise auch Unterstützung für Volltextsuche, die Versionierung von Inhalten und die Unterstützung von SQL als Anfragesprache. Die Schnittstellen zum Zugriff auf ein „Content Repository“ in Java wurden unter dem Titel „Content Repository for Java Technology API“ als „Java Specification Request“ (JSR) 170 (ursprüngliche Version) und 283 (Version 2.0) standardisiert. Letzterer befindet sich allerdings noch im Entwurfsstadium und wird deshalb von Jackrabbit noch nicht vollständig implementiert.

Bei Jackrabbit handelt es sich nicht um ein komplettes Speichersystem im Sinne einer Datenbank, sondern um eine Abstraktionsschicht, die Daten als hierarchische Baumstruktur von Knoten und Attributen (nicht unähnlich zu XML) verwaltet, wobei Daten nur in Attributen gespeichert werden. Jeder Knoten hat einen primären Typ, der bestimmt welche Kindknoten und Attribute erlaubt sind bzw. vorhanden sein müssen.

Zusätzlich ist es durch Mixin-Typen möglich einem Knoten weitere Eigenschaft zuzuweisen. Hat ein Knoten beispielsweise den primären Typ `Person`, der Attribute wie `firstName` und `lastName` definiert, und handelt es sich bei der betreffenden Person um einen Arzt, so könnte der Knoten durch zuweisen des Mixin-Typs `Doctor` ein Attribut `Speciality` erhalten, welches die Fachrichtung angibt.

Die letztendliche Persistierung der Daten überlässt Jackrabbit der darunter liegenden Schicht. Hierbei kann es sich um das Dateisystem, XML-Dateien oder eine XML- bzw. relationale Datenbank handeln. Prinzipiell ist es aber möglich Jackrabbit beliebige Speichersysteme „unterzuschieben“.

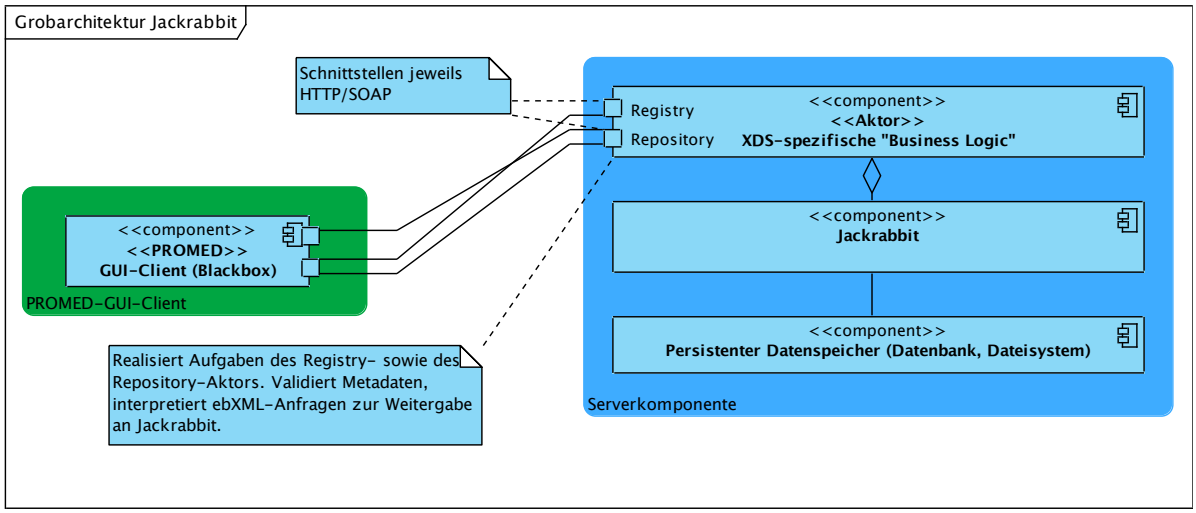

Abbildung 5.2.: Realisierung der Serverkomponente mit Jackrabbit

Für den im Zusammenhang mit dieser Arbeit vorgesehenen Einsatzzweck zur Speicherung der Metadaten als Teil des Registry-Aktors würde sich Jackrabbit auf Grund seiner Fokussierung auf hierarchische Daten gut eignen, auch wenn die Hierarchie der XDS-Daten eher flach ist. Jackrabbit soll dafür als Schicht zwischen der XDS-spezifischen „Business Logic“ und dem persistenten Datenspeicher zum Einsatz kommen (siehe Abbildung 5.2). Nun muss die Business-Logic-Schicht die ankommenden ebXML-Anfragen auswerten und die Metadaten in entsprechende, selbst-definierte Jackrabbit-Knotentypen umwandeln und an Jackrabbit weitergeben. Suchanfragen müssen ebenfalls entsprechend konvertiert werden.

Es wäre also notwendig, einen Teil der ebRS-Spezifikation zu implementieren (zumindest die `Submit`-Methode des Life-Cycle-Managers und den kompletten Query-Manager), was wiederum bedeutet, ein System zu implementieren, das große Teile der ebRIM-Spezifikation bzw. von ebXML im Allgemeinen umsetzt, um die entsprechenden Anfragen interpretieren zu können. Zudem wäre es nötig eine Komponente zu implementieren, die die ebRS-SQL-Anfragen in Jackrabbit-SQL-Anfragen übersetzt, um Suchanfragen

überhaupt verarbeiten zu können und so alle Anfragen des minimalen Anfragekatalogs zu unterstützen.

5.2.2. Realisierung mit freebXML

In Anbetracht des Aufwandes der für eine Implementierung des Serversubsystems auf Basis von Jackrabbit notwendig wäre, wurde von dieser Lösung zugunsten der in diesem Abschnitt beschriebenen Variante der Realisierung mit freebXML abgesehen. FreebXML stellt freie (im Sinn von open-source) Implementierungen für bestimmte ebXML-Technologien bereit. Unter anderem existiert auch eine ebXML-Registry-Implementierung in Java namens „OMAR“, die aktuell in Version 3.1 vorliegt. OMAR setzt die Installation in einem Servlet-Container voraus (z.B. Tomcat) und arbeitet prinzipiell mit allen Datenbanken zusammen, die den SQL-92-Standard unterstützen; getestet wurde laut Entwicklern die Zusammenarbeit mit Derby und HSQLDB, die in der Projektdistribution enthalten sind sowie PostgreSQL und Oracle 9i. Des Weiteren enthält die Distribution ein Programm namens Registry-Browser sowie ein Web-Frontend, welche es erlauben, den Inhalt einer ebXML-Registry zu durchsuchen und neue Elemente in der Registry anzulegen.

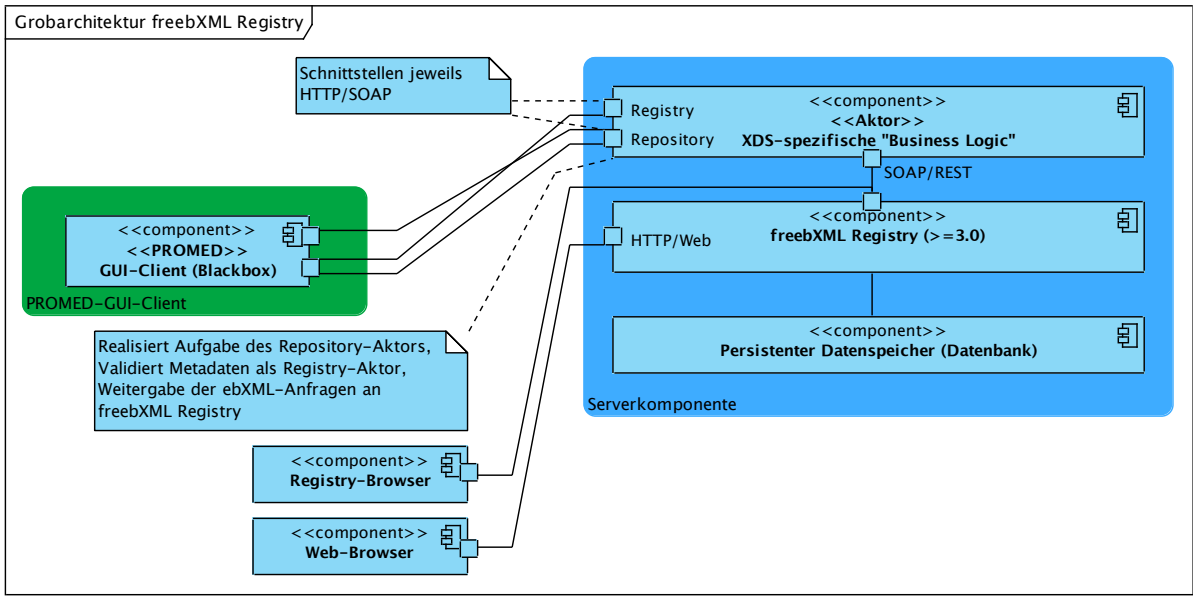

Abbildung 5.3.: Realisierung der Serverkomponente mit der freebXML-Registry

Diese Variante der Realisierung sieht vor, OMAR statt Jackrabbit als Zugriffsschicht zwischen der Business-Logik und einer Datenbank einzusetzen (Abbildung 5.3). Der Vorteil ist, dass die Business-Logik nun nicht mehr in der Lage sein muss die ebXML-codierten Anfragen interpretieren zu können. Es reicht aus, diese an die ebXML-Registry

weiterzureichen und die entsprechende Antwort an den Client zurückzuliefern. Das XDS-spezifische Überprüfen der Metadaten ist technisch relativ einfach möglich, zum Beispiel über Validieren der ebXML-Anfrage gegen ein XML-Schema, sodass auch hierfür keine spezielle Kenntnis von ebXML durch die Business-Logik-Schicht notwendig ist. Da OMAR SQL-Suchanfragen gemäß ebRS-Standard verarbeiten kann, unterstützt die Registry dadurch automatisch alle Anfragen des minimalen Anfragekatalogs von XDS.

Es ist jedoch notwendig die Registry entsprechend zu initialisieren, d.h. die von XDS verwendeten Datentypen in der Registry anzulegen; siehe hierfür Kapitel 3.1.

5.2.3. Realisierung mit NIST-Komponenten

Trotz der Erleichterungen die der Einsatz der freebXML-Variante mit sich bringt, ist einiger Aufwand nötig, um die Schnittstellen und Geschäftslogik für XDS zu entwickeln. Dies hat letztendlich dazu geführt, dass das Serversubsystem mit Hilfe der Komponenten von NIST aufgebaut wurde.

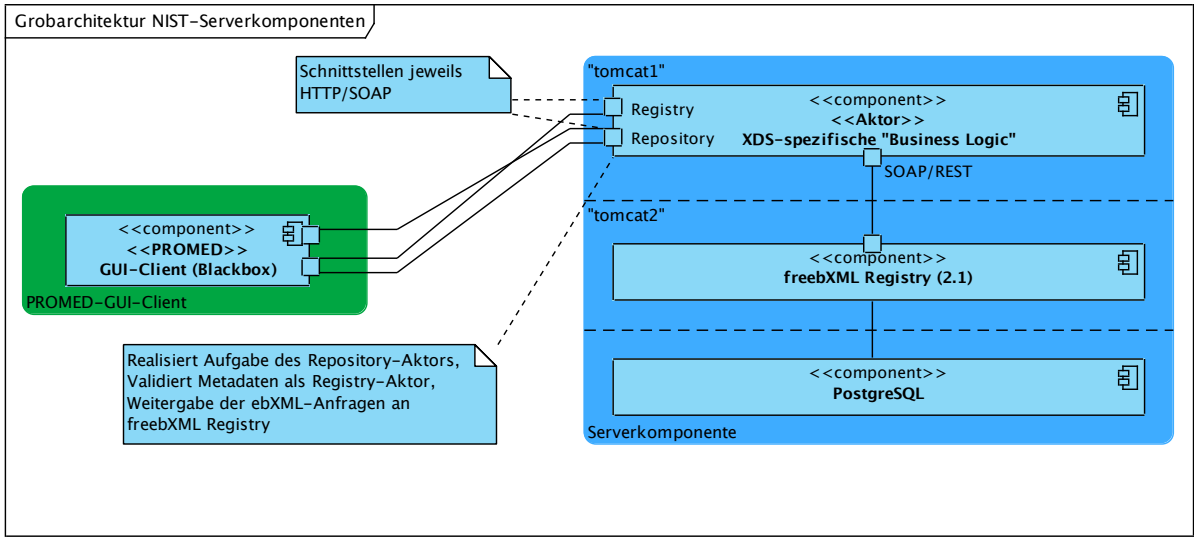

Abbildung 5.4.: Realisierung der Serverkomponente mit NIST-Komponenten

Der Ansatz der NIST (Abbildung 5.4) ist abgesehen von Versionsunterschieden sehr ähnlich zu der im vorherigen Abschnitt beschriebenen Lösung. Auch die NIST-Implementierung setzt auf die freebXML-Registry, allerdings in Version 2.1, welche nicht mehr frei verfügbar, jedoch im Paket der NIST-Komponenten enthalten ist. OMAR setzt dabei auf eine PostgreSQL-Datenbank auf, deren Schema und initialer Datenbestand ebenfalls im Softwarepaket enthalten ist. Die Realisierung der XDS-Transaktions-Schnittstellen und die Business-Logik-Schicht basiert auf Apache Axis2, einem Framework mit dem sich SOAP-basierte Web-Services realisieren lassen. Die Komponenten laufen dabei auf zwei Tomcat-Servlet-Container verteilt. „tomcat2“ ist für den Betrieb der OMAR-Instanz

zuständig, während in „tomcat1“ die Axis2-Schnittstellen laufen. Zusätzlich beinhaltet das Paket einen Patient-Identity-Source-Aktor, der gültige Patienten-IDs für den XDS-Registry-Aktor zur Verfügung stellt. Neue Patienten-IDs lassen sich dabei bequem über ein Web-Interface generieren und sind dann sofort verwendbar.

Die XDS-spezifische Überprüfung der Metadaten lässt sich über eine XML-Datei relativ frei konfigurieren. So ist es unter anderem möglich, eine bestimmte Menge an erlaubten Werten für die einzelnen Codierungs-Attribute (zum Beispiel `formatCode`) der XDS-Datentypen vorzugeben, auf deren Einhaltung der Aktor dann alle eingehenden Metadaten entsprechend prüft.

6. Systementwurf

6.1. GUI-Client

Der GUI-Client wird in Java entwickelt, um die Anwendung unabhängig von einer bestimmten Plattform zu halten und um die Verwendung der OHF-Komponente zu erleichtern. Um für die Zukunft sowohl eine leichte Weiterentwickelbarkeit des Clients, als auch eine Austauschbarkeit der XDS-Implementierung zu garantieren, stand eine möglichst lose Kopplung aller verwendeter Komponenten im Vordergrund. Dies führte zu einer Reihe von Entwurfsentscheidungen, die in den folgenden Kapiteln näher erläutert werden.

6.1.1. Schnittstellen zu OHF

Die relativ technischen Schnittstellen, die OHF zur Verfügung stellt sowie der Wunsch, den Client möglichst unabhängig von einer speziellen OHF-Implementierung zu halten, führte zum Entwurf zweier eigener Schnittstellen, die jeweils die Funktionen des Source- und Consumer-Aktors abbilden. Die Vermittlung zwischen diesen Client-spezifischen, angeforderten Schnittstellen und den von OHF angebotenen Schnittstellen vermitteln zwei Adapter-Komponenten (vgl. [GHJV95], S. 139 und [Sie04], S. 68). So lassen sich durch Austausch der Adapter auch die zu Grunde liegenden Aktor-Implementierungen von OHF durch andere ersetzen.

Abbildung 6.1 zeigt oben die angeforderte Schnittstelle `XDSConsumerAdaptor`, unten die durch OHF angebotene Schnittstelle `Consumer` sowie die Adapter-Komponente (mittig im Package `impl`), die zwischen beiden vermittelt, indem sie die angeforderte Schnittstelle implementiert und die angebotene Schnittstelle der Fremd-Komponente aggregiert.

Um weitere Unabhängigkeit zu schaffen, kommen client-seitig bis zum Adapter keine OHF-spezifischen Datenobjekte zur Codierung der Metadaten zum Einsatz bzw. werden diese in Transferobjekten gekapselt. Unterstützt eine neue Implementierung keine OHF-Datenobjekte, so lässt sich dies durch Neuimplementierung der Transferobjekte bzw. notfalls auch durch Transformation der Daten im Adapter beheben.

6.1.2. GUI-Entwurfsmuster

Auch bei der Entwicklung der eigentlichen GUI-Anwendung war eine möglichst geringe Kopplung der Komponenten in Hinsicht auf eine leichte Erweiterbarkeit wich-

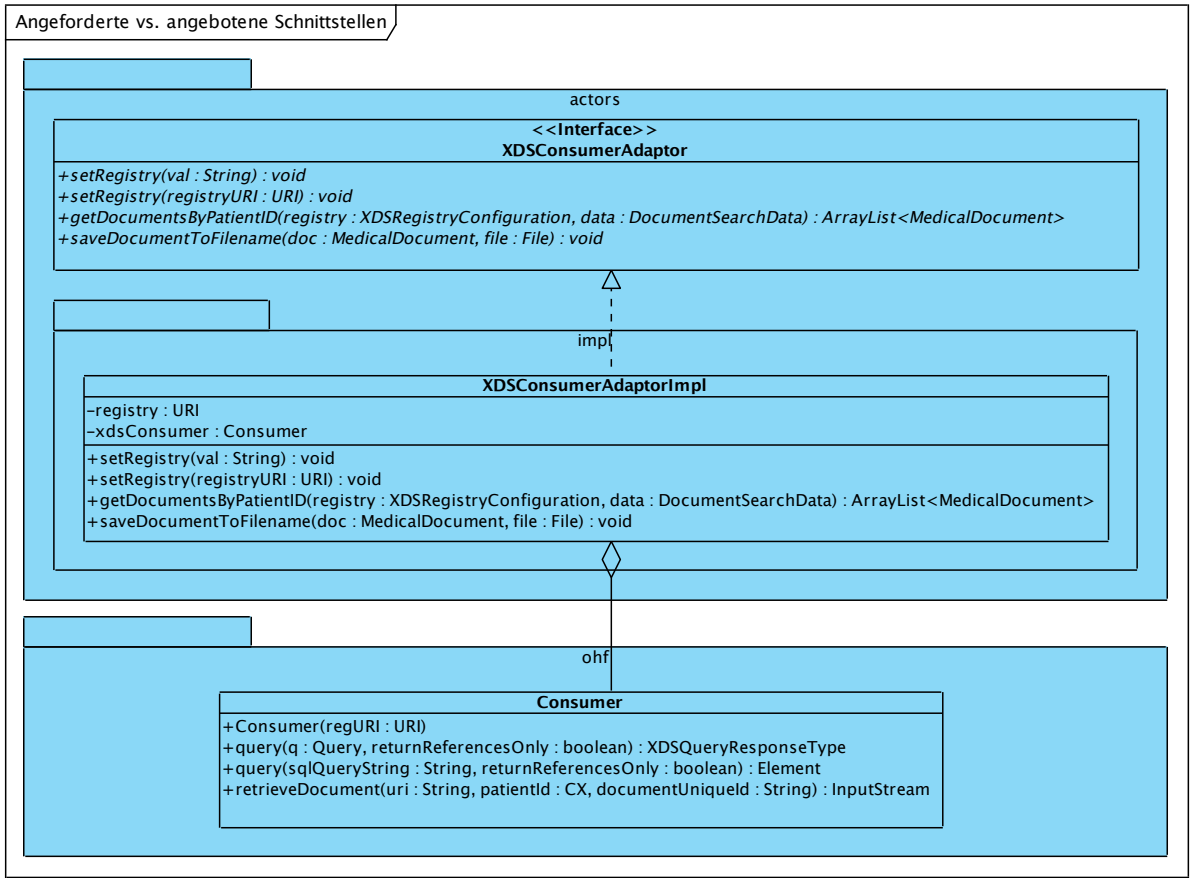

Abbildung 6.1.: Überbrückung zwischen angeforderter und angebotener Schnittstelle mit Hilfe eines Adapter am Beispiel der XDS-Consumer-Komponente

tig. Grundsätzlich orientiert sich die Architektur des GUI-Client in dieser Hinsicht am Model-View-Controller-Muster (MVC, vgl. [Fow03], S. 125). Die Umsetzung wurde jedoch variiert, um den Aufwand der Implementierung einigermaßen in normaler Relation zu Größe und Funktionsumfang zu halten. MVC bedeutet prinzipiell die Einführung von drei Komponenten, nämlich dem Model, das die eigentliche Geschäftslogik enthält, dem View, also der Anzeigekomponente, und dem Controller, der den Kontrollfluss der Anwendung steuert, indem er auf Grund des Zustands der Anwendung und dem aufgetretenen Ereignis (in der Regel eine Benutzereingabe, wie das Anklicken einer Schaltfläche) bestimmte Funktionen ausführt. Theoretisch wären nun also Event-Objekte nötig, die bestimmte auftretende Ereignisse codieren, sodass beispielsweise die View-Komponente ein ganz bestimmtes Event-Objekt an den Controller übergibt, wenn der Benutzer eine bestimmte Schaltfläche drückt. Da der GUI-Client Swing bzw. den AWT-Framework zur Realisierung der Anzeigen einsetzt, wäre es naheliegend, die in diesem Framework bereits vorhandenen Event-Objekte auch für die Signalisierung in Richtung Anwendungs-

controller zu verwenden. Der AWT-Framework stellt eine Basisklasse zur Verfügung, die bereits bestimmte Attribute enthält. Die Event-Objekte für die verschiedenen Ereignisse und Komponenten, die diese auslösen, sind Instanzen von Klassen, die von dieser Basisklasse abgeleitet sind. So erzeugt beispielsweise das Anklicken einer Schaltfläche eine Event-Objekt-Instanz einer anderen Klasse, als das Anwählen eines Elements in einer Listenansicht. Gemeinsam sollte aber allen Klassen sein, dass sie von der Basisklasse `AWTEvent` abgeleitet sind und somit alle eine numerische ID besitzen, die das Ereignis identifiziert. Nun ist es aber leider so, dass die Klassen bestimmter Ereignisse bestimmter Elemente nicht von der Klasse `AWTEvent` abgeleitet sind, sondern von deren Basisklasse `EventObject`, welche über keinen numerischen Identifikator verfügt. Somit ist im Controller nicht mehr einfach feststellbar, welches Ereignis aufgetreten ist und welche Funktion nun ausgeführt werden sollte. Es wäre nun also nötig, eigene Event-Klassen zu implementieren, die jeweils bestimmte Ereignisse darstellen, was eine erhebliche Anzahl an Klassen bedeuten würde. In Anbetracht des geringen Funktionsumfangs des GUI-Clients wurde deshalb entschieden, dass die View-Komponente direkt, je nach Ereignis, eine bestimmte Funktion des Controllers aufruft, anstatt ein Event-Objekt an diesen zu übergeben. So ist der bei einer Aktion auszuführende Code nach wie vor beim Controller hinterlegt, jedoch findet das „Wiring“, also welche Funktion am Controller aufgerufen werden soll, in der View-Komponente statt.

Die eigentliche Geschäftslogik wurde zur leichteren Austausch- und Erweiterbarkeit sowie zur Schaffung der Möglichkeit, die Funktion jeweils in einem eigenen Hintergrundprozess ausführen zu können, über das Command-Muster implementiert (siehe [GHJV95], S. 233). Dies bedeutet, dass die eigentliche Funktionalität jeweils in einer spezialisierten Klasse ausprogrammiert ist, wobei alle Klassen eine gemeinsame Basisklasse besitzen bzw. eine gemeinsame Schnittstelle implementieren und die Ausführung unabhängig von der konkreten Klasse über eine gleichlautende Methode anzustoßen ist.

6.2. Remodularisierung von OHF

Da OHF unter anderem zur Abbildung der XDS-Metadaten auf den Eclipse-Modeling-Framework zurückgreift, bestehen natürlich einige Abhängigkeiten zu dessen Implementierung. Ebenso bestehen Abhängigkeiten zu weiteren Komponenten von OHF. Diese Abhängigkeiten haben ihrerseits wieder Abhängigkeiten zu Paketen aus OHF und EMF. Zusätzlich bestehen weitere Abhängigkeiten zu Komponenten zur XML-Verarbeitung und zu einer Logger-Komponente. Dies macht es einerseits schon schwierig, die zum Betrieb beider Aktoren nötigen Abhängigkeiten zu bestimmten. Andererseits ist es fraglich, ob angesichts der Vielzahl an Abhängigkeiten die Herauslösung eines Aktors die Zahl der Abhängigkeiten überhaupt signifikant reduziert.

7. Evaluation

7.1. Zusammenfassung der Ergebnisse

Diese Arbeit hat Möglichkeiten zum Entwurf und zur Realisierung eines exemplarischen XDS-Systems, welches zum Test und zur Evaluation von XDS-Komponenten einsetzbar ist, aufgezeigt. Nach Erörterung des XDS-Profils und der grundlegenden Technologien, die im Rahmen von XDS zum Einsatz kommen, wurden die Anforderungen an ein XDS-System gemäß Spezifikation zusammengestellt. Zusätzlich wurden die speziellen Anforderungen der Testumgebung spezifiziert.

An Hand dieser Anforderungsdefinition wurde das „Open Health Framework“ auf Umfang und Tauglichkeit hin überprüft. Dieser Framework erwies sich grundsätzlich als tauglich und unterstützt weitgehend alle definierten Anforderungen des XDS-Profils. Bei der Evaluation stellte sich jedoch heraus, dass nur Implementierungen der Clientseitigen Aktoren des XDS-Profils in OHF enthalten sind. Daraufhin wurden mehrere Entwurfsalternativen für die Serverkomponente ausgearbeitet.

Der Einsatz von Jackrabbit schien viel versprechend im Hinblick auf die Unterstützung hierarchischer Daten. Auf Grund des hohen Aufwands, der in die Entwicklung des notwendigen Adapters zur Umsetzung der ebXML-Anfragen auf Jackrabbit hätte investiert werden müssen, wurde diese Alternative nicht realisiert.

Der darauf folgende Ansatz, eine fertige ebXML-Registry-Implementierung einzusetzen, erwies sich als erfolgreicher. Letztendlich wurde auf die frei verfügbaren Komponenten der NIST zurückgegriffen, die einen XDS-Registry- und XDS-Repository-Aktor enthalten, deren Architektur nach genau diesem Ansatz entworfen ist.

Zusammen mit dem GUI-Client auf Basis von OHF, dessen Anforderungen und dessen Entwurf ebenfalls als Teil dieser Arbeit dokumentiert sind, ergibt sich so ein funktionsfähiges XDS-System, mit dem weitere Komponenten anderer Hersteller getestet werden können.

7.2. Weiterführende Aufgaben

Das im Rahmen dieser Arbeit entstandene XDS-System stellt die grundlegenden Funktionen zum Austausch von Dokumenten bereit. So lassen sich Dokumente in das zentrale Repository einstellen sowie per Suchanfragen an den Registry-Aktor auffinden und wieder vom Repository-Aktor abrufen. Das XDS-Profil definiert darüber hinaus jedoch

noch weitere Funktionen und Elemente. So bietet der GUI-Client momentan beispielsweise nicht die Möglichkeit, mehrere Dokumente pro Vorgang abzuschicken, Dokumente in Ordnern zu organisieren oder bereits im System vorhandene Dokumente zu referenzieren. Auch lässt sich momentan nur nach Dokumenten an Hand der Patienten-ID suchen. Auf längere Sicht wäre es interessant hier weitere Funktionen in den GUI-Client zu integrieren.

Auf Seiten der Serverkomponente könnte die Implementierung der NIST einer genaueren Untersuchung hinsichtlich ihrer Modularisierbarkeit unterzogen werden, um die Aktoren zum Beispiel auf getrennten Knoten zu betreiben oder um den Einsatz mehrerer Repository-Aktoren zu ermöglichen.

A. Handbuch GUI-Client

A.1. Installation

Zur Installation des GUI-Clients genügt es, die komprimierte Archivdatei auszupacken und die Datei `gui-client.bat` unter Windows bzw. `gui-client.sh` unter Linux auszuführen. Der GUI-Client benötigt mindestens Java 1.6, alle sonstigen Abhängigkeiten sind im Paket enthalten.

A.2. Verwendung des GUI-Clients

A.2.1. Einstellen von Dokumenten

Nach dem Starten der Anwendung ist automatisch der Tab zum Einstellen von Dokumenten ausgewählt. Die Eingabe der Daten beginnt mit der Auswahl des Repository-Aktors in dem das Dokument gespeichert werden soll. Hierzu wird eine Adresse aus Liste 1 in Abblidung A.1 ausgewählt. Alternativ kann auch eine gültige URI eines Repository-Aktors in das Feld eingegeben werden. Danach wird der Patient durch Auswahl der gewünschten Patienten-ID aus Liste 2 (Abbildung A.1) bestimmt. Auch hier ist die Eingabe einer nicht in der Liste enthaltenen ID möglich. Es ist jedoch zu beachten, dass in diesem Fall die komplette ID einschließlich OID der Affinity-Domain und Typ der ID angegeben werden muss (z.B. `abc123^^^&1.2.3.4.5&ISO` statt nur `abc123`). Danach könne Titel und Format des Submission-Sets (Abbildung A.1, Punkt 3) gewählt werden. Zuletzt müssen noch das lokale Dokument sowie dessen Titel und weitere Eigenschaften ausgewählt werden (Abbildung A.1, Punkt 4).

Sind alle notwendigen Daten vorhanden, wird die *Submit*-Schaltfläche (Abbildung A.2, Punkt 1) anwählbar. Mit einem Klick auf diese wird der Einstell-Vorgang gestartet. Nach erfolgreicher Übertragung des Dokuments und der Metadaten erscheint eine entsprechende Dialogbox (Abbildung A.3). Sollte während der Übermittlung ein Fehler eintreten, wird der Vorgang abgebrochen und es erscheint ebenfalls eine Dialogbox mit einer Fehlerbeschreibung.

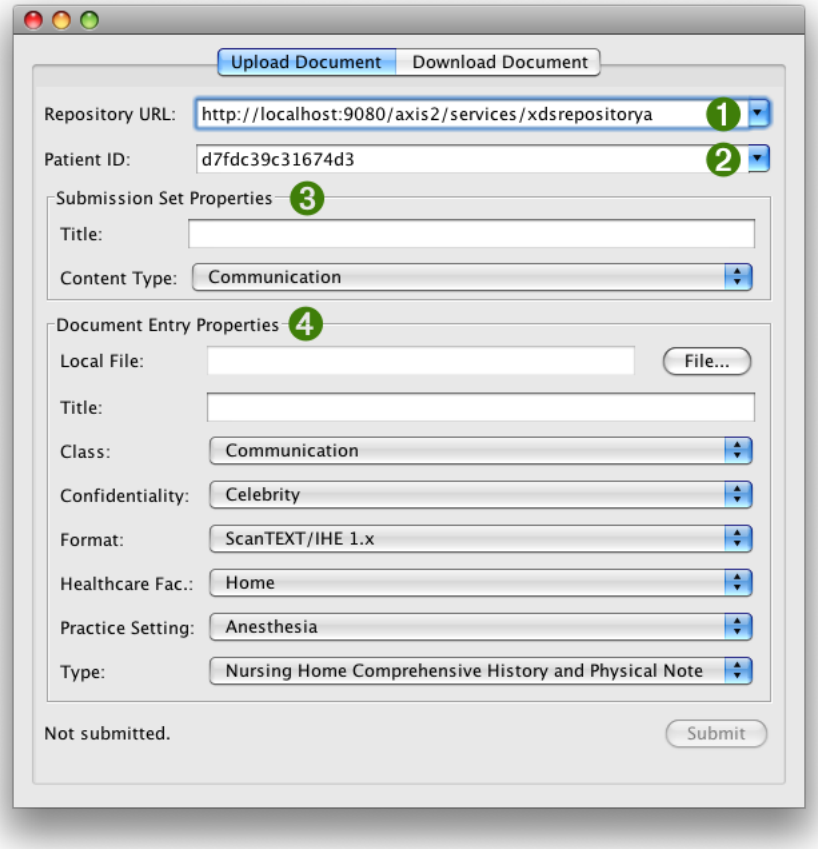

Abbildung A.1.: Einstellen von Dokumenten 1 - Eingabe der Daten

A.2.2. Suchen nach Dokumenten

Nach dem Starten der Anwendung ist automatisch der Tab zum Einstellen von Dokumenten ausgewählt. Um zur Suchmaske zu gelangen muss der Tab „Download Document“ (Abbildung A.4, Punkt 1) durch einen Klick ausgewählt werden.

Im folgenden Dialog können nun die Suchparameter bestimmt werden. Hierfür muss zuerst der gewünschte Registry-Aktor aus der List ausgewählt werden. Alternativ ist es auch hier möglich die Adresse eines Aktors in Form einer URI anzugeben (Abbildung A.5, Punkt 1). Danach wird der Patient festgelegt, nach dessen Dokumenten gesucht werden soll. Hierfür kann entweder die entsprechende ID aus der Liste (Abbildung A.5, Punkt 2) ausgewählt oder eine ID direkt angegeben werden. Auch hier ist zu beachten, dass eine vollständige Patienten-ID angegeben werden muss (siehe Kapitel A.2.1). Ein Klick auf die Schaltfläche „Search“ (Abbildung A.5, Punkt 3) startet den Suchvorgang.

Nach Abschluss des Suchvorgangs werden etwaige Suchergebnisse in Liste 1 (Abbildung A.6) angezeigt. Jeder Eintrag in dieser Liste entspricht einem gefundenen Do-

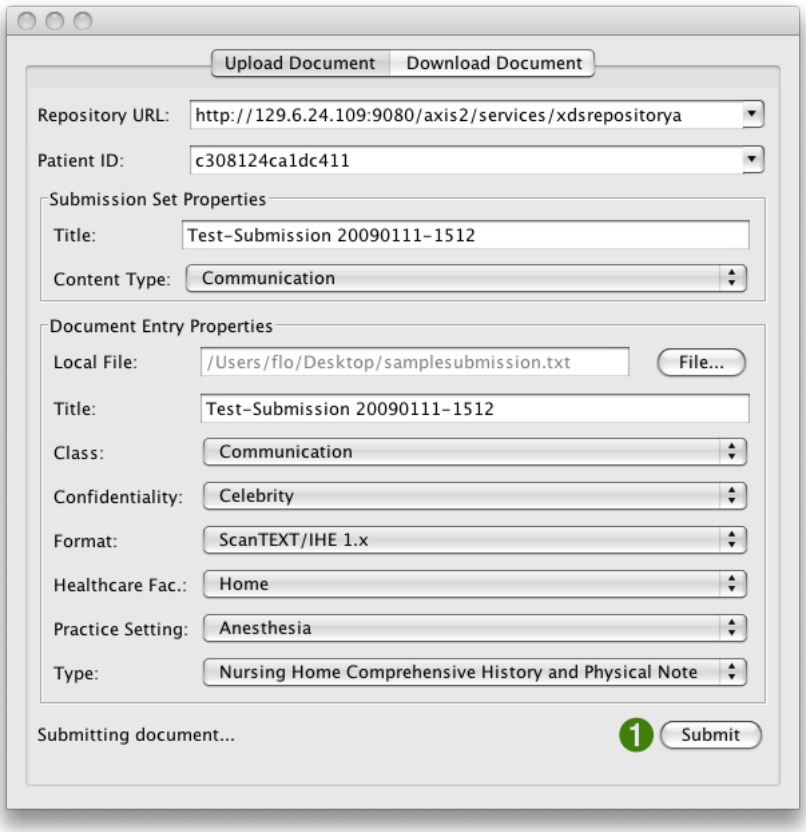

Abbildung A.2.: Einstellen von Dokumenten 2 - Absenden der Daten

kument. Weitere Eigenschaften jedes Dokuments können unterhalb der Liste (Abbildung A.7, Punkt 2) angezeigt werden, indem ein Dokument in der Liste selektiert wird.

Ein neuer Suchvorgang kann jederzeit nach Veränderung der Suchparameter über die Schaltfläche „Search“ gestartet werden.

A.2.3. Abrufen von Dokumenten

Das Abrufen von Dokumenten setzt voraus, dass vorher ein Suchvorgang ausgeführt wurde (siehe Kapitel A.2.2). Wird in der Liste der Suchergebnisse (Abbildung A.8, Punkt 1) ein Dokument selektiert, wird auch die Schaltfläche „Download Document“ anwählbar (Abbildung A.8, Punkt 2). Ein Klick auf diese führt zum Auswahldialog in Abbildung A.9. Hier kann der Speicherort des Dokuments im lokalen Dateisystem bestimmt werden. Ein Klick auf die Schaltfläche „Sichern“ (Abbildung A.9, Punkt 2) schließt den Dialog und startet den Downloadvorgang, über dessen Abschluss ebenfalls ein Dialog informiert (Abbildung A.10).

Danach ist das Dokument im lokalen Dateisystem abgelegt.

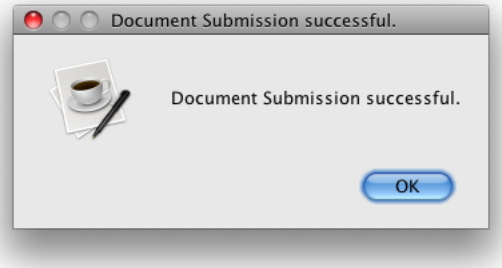

Abbildung A.3.: Einstellen von Dokumenten 3 - Ergebnis

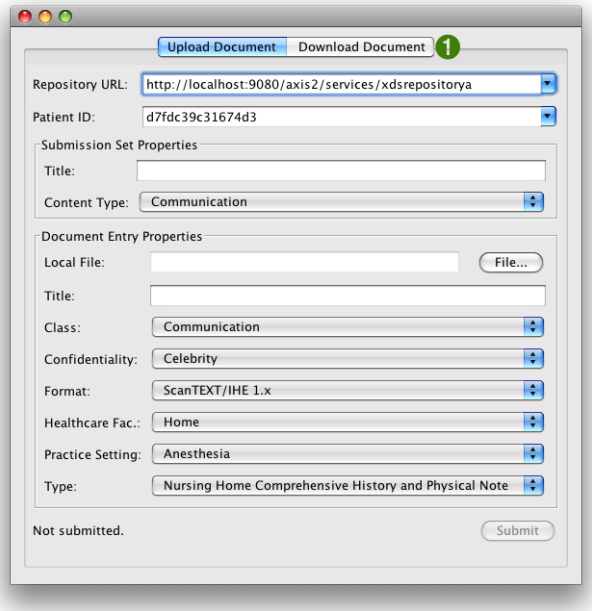

Abbildung A.4.: Suchen nach Dokumenten 1 - Auswahl des Suchdialogs

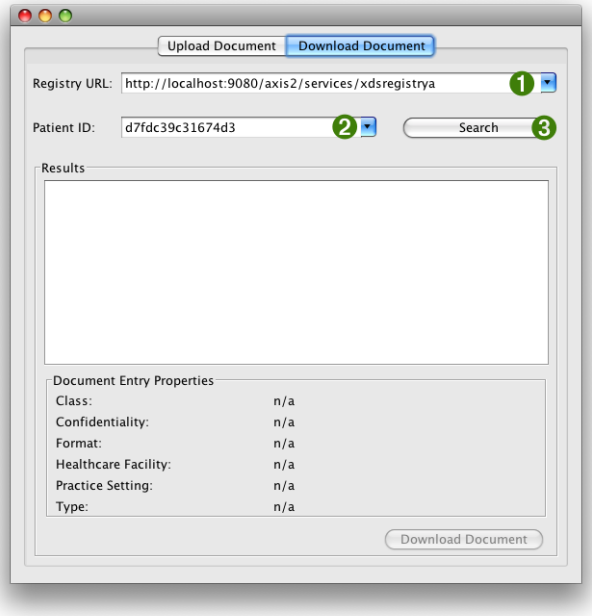

Abbildung A.5.: Suchen nach Dokumenten 2 - Auswahl der Suchparameter

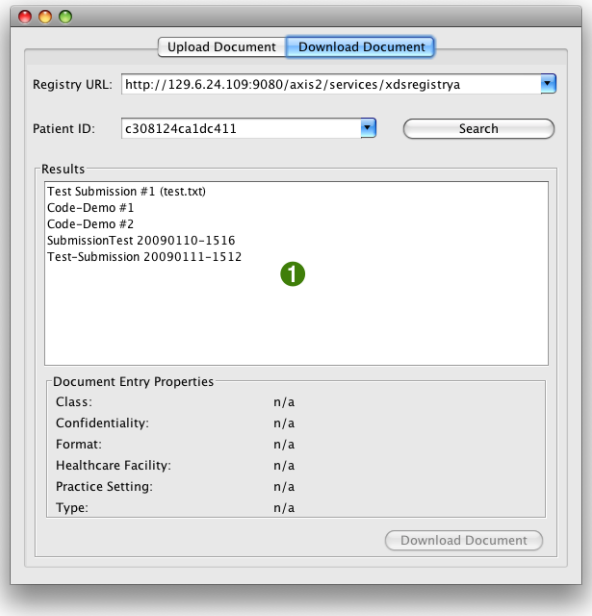

Abbildung A.6.: Suchen nach Dokumenten 3 - Anzeige der Suchergebnisse

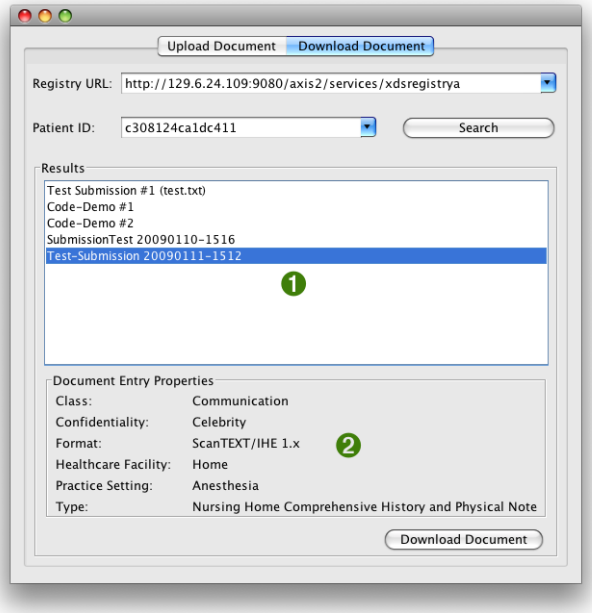

Abbildung A.7.: Suchen nach Dokumenten 4 - Anzeige weiterer Dokumenteigenschaften

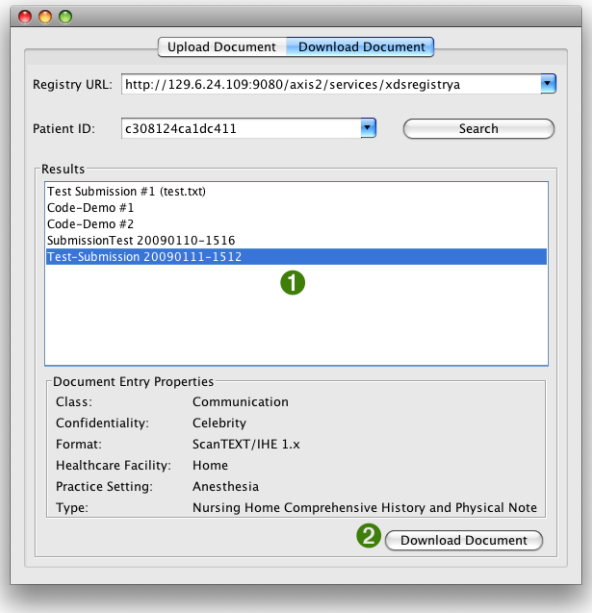

Abbildung A.8.: Abrufen von Dokumenten 1 - Auswahl des Dokuments

Abbildung A.9.: Abrufen von Dokumenten 2 - Auswahl des Speicherorts

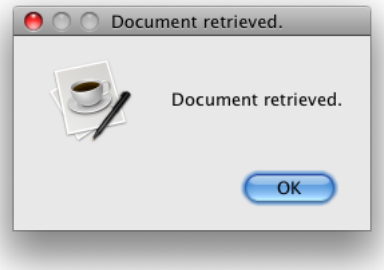

Abbildung A.10.: Abrufen von Dokumenten 3 - Ergebnis

B. Installation der NIST-Komponenten

B.1. Voraussetzungen

Für die Installation der NIST-Komponenten wird eine PostgreSQL-Datenbank sowie der Java 5 JDK benötigt. Alle weiteren Abhängigkeiten sind in den Installationsarchiven enthalten.

B.2. Installationsprozess

Der eigentlichen Installationsprozess wird in der Anleitung der NIST ([nisrz]) sehr detailliert und nachvollziehbar beschrieben, weshalb hier nur eine Anmerkung zum Installationsort folgt.

Die beiden Tomcat-Server der NIST müssen zwingend unter `/usr/local/tomcat1` und `/usr/local/tomcat2` verfügbar sein. Es ist laut [nisrz] wegen Unterschieden in den Versionen der Abhängigkeiten auch nicht möglich diese zu einer Instanz zusammenzufassen. Es wird trotzdem empfohlen die Tomcat-Instanzen unter `/opt/xdsrig/tomcat_xds` (tomcat1) bzw. `/opt/xdsrig/tomcat_freebxml` (tomcat2) zu installieren und mit folgenden Befehlen Verweise zu diesen Verzeichnissen zu erstellen:

```
cd /usr/local/
ln -s /opt/xdsrig/tomcat_xds tomcat1
ln -s /opt/xdsrig/tomcat_freebxml tomcat2
```

Listing B.1: Anlegen der Verweise

B.3. Start- und Stopskripte

Um nicht beide Tomcat-Server jeweils Starten und Stoppen zu müssen, können folgende Shell-Skripte verwendet werden:

```
#!/bin/bash
# Start both tomcat servers for NIST XDS Registry/Repository
# Author: Florian Wagner, flo@flowagner.org
```

```
# Version: Jan 12, 2009
#
# Server locations are /usr/local/tomcat{1,2}
#
# start tomcat2 first. it contains the OMAR registry instance
cd /usr/local/tomcat2/bin
./startup.sh

# start tomcat1. it contains the xds service endpoints and
   business logic
cd /usr/local/tomcat1/bin
./startup.sh
```

Listing B.2: Skript zum Starten der Server

```
#!/bin/bash
# Stop both tomcat servers for NIST XDS Registry/Repository
# Author: Florian Wagner, flo@flowagner.org
# Version: Jan 12, 2009
#
# Server locations are /usr/local/tomcat{1,2}
#
# stop tomcat1 first.
cd /usr/local/tomcat1/bin
./shutdown.sh

# stop tomcat2
cd /usr/local/tomcat2/bin
./shutdown.sh
```

Listing B.3: Skript zum Stoppen der Server

C. Erstellung von SSL-Zertifikaten

Glossar

Affinity Domain	Eine Gruppe von medizinischen Versorgungseinrichtungen, die auf der Basis von gemeinsamen Richtlinien zusammenarbeiten und sich gegebenenfalls auch Infrastruktur teilen, zum Beispiel ein Krankenhausverbund, Abteilungen innerhalb eines Krankenhauses oder mehrere Krankenhäuser und niedergelassene Ärzte innerhalb einer Region.
ATNA	*Audit Trail and Node Authentication*, ein IHE-Profil.
DICOM	*Digital Imaging and Communications in Medicine*. Ein medizinisches Bildformat.
ebXML	*Electronic Business using eXtensible Markup Language*, `http://www.ebxml.org/`
ebXML RIM	*ebXML Registry Information Model*
ebXML RS	*ebXML Registry Services*
HL7 CDA	*HL7 Clinical Document Architecture*. XML-basierte Auszeichnungssprache zur Beschreibung von medizinischen Dokumenten.
IHE	'Integrating the Health Enterprise' ist eine Initiative von Medizinern und Wirtschaft mit dem Ziel, die Zusammenarbeit zwischen Computersystemen im medizinischen Umfeld, z.B. durch die Einführung von Standards wie HL7 oder DICOM, zu verbessern. `http://www.ihe.net`
NIST	National Institute for Standards and Technology
OASIS	*Organization for the Advancement of Structured Information Standards*, `http://www.oasis-open.org/`
OHF	Open Health Framework
OHT	Open Health Tools
OID	Object Identifier
PDQ	*Patient Demographics Query*, ein IHE-Profil.
PIX	*Patient Identifier Cross-Referencing*, ein IHE-Profil.
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
UN/CEFACT	*United Nations Centre for Trade Facilitation and Electronic Business*, `http://www.unece.org/`
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
XDS	*Cross-Enterprise Document Sharing*, ein IHE-Profil.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ebr01a] *OASIS/ebXML Registry Information Model v2.0 Approved Committee Specification.* `http://www.ebxml.org/specs/ebrim2.pdf`. Version: Dezember 2001, Abruf: 5. Januar 2009

[ebr01b] *OASIS/ebXML Registry Information Model v2.0 Approved Committee Specification.* `http://www.ebxml.org/specs/ebrs2.pdf`. Version: Dezember 2001, Abruf: 5. Januar 2009

[Fow03] Fowler, Martin: *Patterns of enterprise application architecture.* Boston : Addison-Wesley, 2003. – ISBN 0321127420 (alk. paper)

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns: elements of reusable object-oriented software.* Reading, Mass. : Addison-Wesley, 1995. – ISBN 0201633612 (acid–free paper)

[iti08a] *IHE IT Infrastructure (ITI) Technical Framework, vol. 1 (ITI TF-1): Integration Profiles, Rev 5.1 November 14, 2008.* `http://www.ihe.net/Technical_Framework/upload/IHE_ITI_TF_5-0_Vol1_FT_2008-11-14.pdf`. Version: November 2008, Abruf: 28. Dezember 2008

[iti08b] *IHE IT Infrastructure (ITI) Technical Framework, vol. 2 (ITI TF-2): Transactions, Rev 5.1 November 14, 2008.* `http://www.ihe.net/Technical_Framework/upload/IHE_ITI_TF_5-0_Vol2_FT_2008-11-14.pdf`. Version: November 2008, Abruf: 28. Dezember 2008

[NIS08] NIST: *About NIST.* `http://www.nist.gov/public_affairs/nandyou.htm`. Version: April 2008, Abruf: 30. Dezember 2008

[nisrz] *NIST Registry/Repository Installation instructions.* `http://129.6.24.109:9080/XdsDocs/opensource/install.pdf`. Version: 2008 März, Abruf: 12. Januar 2009

[Sie04] Siedersleben, J.: *Moderne Softwarearchitektur.* dpunkt-Verl., 2004

[xds07a] *Notes on XDS Profile - Sample Query.* `http://ihewiki.wustl.edu/wiki/index.php/Notes_on_XDS_Profile#Sample_Query`. Version: Dezember 2007, Abruf: 30. Dezember 2008

[xds07b] *Notes on XDS Profile - UUIDs Definied by XDS*. http://ihewiki.wustl.edu/wiki/index.php/Notes_on_XDS_Profile#UUIDs_Defined_by_XDS. Version: Dezember 2007, Abruf: 30. Dezember 2008

[xds07c] *Notes on XDS Profile - XDS Provide and Register Document Set transaction*. http://ihewiki.wustl.edu/wiki/index.php/Notes_on_XDS_Profile#XDS_Provide_and_Registry_Document_Set_transaction. Version: Dezember 2007, Abruf: 30. Dezember 2008