Webanwendungsentwicklung mit XML-Techniken

Christian Forster

<application type="web"/>

Westfälische
Wilhelms-Universität
Münster

DBIS Group
Databases &
Information Systems

Webanwendungsentwicklung mit XML-Techniken

Inauguraldissertation
zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften durch die
Wirtschaftswissenschaftliche Fakultät der
Westfälischen Wilhelms-Universität Münster

Vorgelegt von

Diplom-Wirtschaftsinformatiker
Christian Forster
aus Essen

Februar 2014

Dekan	Prof. Dr. Christoph Watrin, StB
Erster Gutachter	Prof. Dr. Gottfried Vossen
Zweiter Gutachter	Prof. Dr. Herbert Kuchen
Mündliche Prüfung	24. April 2014

Geleitwort

Software-Entwicklung ist eine zentrale Aufgabe der modernen Informatik, mit zahlreichen Anwendungen in nahezu allen Bereichen des täglichen Lebens. Sie kann bereits auf eine gewisse Historie sowie innerhalb der Informatik auf diverse Paradigmenwechsel zurückblicken, gehört aber dennoch zu den nach wie vor „schwierigen" Teilen der Entwicklung eines Informatik-Systems. Belegt wird dies u. a. dadurch, dass größere Desaster im Kontext von Informatik-Systemen typischerweise durch Software-Fehler (und seltener durch Hardware-Fehler) verursacht werden. Durch die rasante Verbreitung des Internet und seines zentralen Dienstes (World-Wide) Web steht heute vielfach die Entwicklung von Web-Anwendungen im Vordergrund, also von Programmen, die ihre Benutzerschnittstelle mit Web-Techniken realisieren und die über ein Netzwerk bereitgestellt werden. Dies ist sowohl für Anwender als auch für Entwickler attraktiv: für Anwender, weil Web-Anwendungen nicht clientseitig installiert oder gewartet werden müssen, da sie beim Aufruf der entsprechenden Web-Adresse einfach geladen werden; für Entwickler, weil eine Verwendung der hier häufig zum Einsatz kommenden Skriptsprachen eine Reihe von Vorteilen bringen kann.

Software wird heute typischerweise in Form einer Schichtenarchitektur entwickelt, wobei eine Unterscheidung in die drei Schichten Datenhaltung, Geschäftslogik und Präsentation (von unten nach oben) weit verbreitet ist. Dies ermöglicht u. a. eine klare Aufgabentrennung, die Möglichkeit der Entwicklung einzelner Schichten unabhängig von den anderen sowie die des Austauschs einzelner Schichten ohne Auswirkung auf die angrenzenden, sofern – so könnte man meinen – eine Kommunikation zwischen den Schichten konsequent über klar definierte Schnittstellen erfolgt. Die Praxis zeigt jedoch, dass die Situation keineswegs so einfach ist; in der Tat verwenden insbesondere die untere (Datenhaltungs-) sowie die mittlere (Geschäftslogik-) Schicht meist Paradigmen, zwischen denen ein Übergang nicht leicht zu bewerkstelligen ist. Konkret erfolgt Datenhaltung nach wie vor überwiegend relational, Anwendungsentwicklung dagegen primär objekt-orientiert; bei Web-Anwendungen kommt ferner der Übergang zwischen serverseitiger, meist objekt-orientierter Technik und cli-

entseitiger Web-Technik hinzu. Diese Paradigmenwechsel bzw.-brüche führen zu vermeidbaren Fehlerquellen bzw. zu Fehlern.

Die Arbeit von Christian Forster befasst sich mit einer Überwindung dieses sogenannten Impedance Mismatch, der nicht selten in Software-Architekturen anzutreffen ist und der bis heute keine einheitliche, durchgängig verwendete Lösung erfahren hat. Er schlägt als Abhilfe einen konsequent auf XML-Techniken basierenden Ansatz vor, den er aus den Erfahrungen eines DFG-Projekts, in welchem u. a. Software für eine klinische Anwendung entwickelt wurde, ableitet. Im Rahmen dieses Projektes wurde einerseits ein funktionierender Prototyp entwickelt, andererseits arbeitet Forster hier eine Abstraktion dessen aus, die auf grundlegende, auf andere Anwendungsbereiche übertragbare Prinzipien hinweist, was die Arbeit für Software-Architekten und -Entwickler interessant macht.

Münster, im Mai 2014
Gottfried Vossen

Danksagung

Liebe Leserin, lieber Leser,

meinem Doktorvater Prof. Dr. Gottfried Vossen gebührt Dank für die motivierende Betreuung meiner Dissertation und seinen unterstützenden fachlichen Rat. Für die Möglichkeit, meine Forschung auf Konferenzen vorzustellen und zu diskutieren bin ich sehr dankbar.

Prof. Dr. Herbert Kuchen danke ich für seine Tätigkeit als Zweitgutachter und Prof. Dr. Jens Leker für seine Tätigkeit als weiterer Prüfer bei der Disputation.

Meinen Kollegen am Lehrstuhl für Informatik möchte ich herzlich für die vielfältigen wissenschaftlichen Diskurse und wertvollen persönlichen Erörterungen danken: David Fekete, Dr. Till Haselmann, Iman Kamehkhosh, Dr. Jens Lechtenbörger, Nicolas Pflanzl, Fabian Schomm, Florian Stahl und Dr. Gunnar Thies. Danken möchte ich darüber hinaus für kompetente Unterstützung in technischen Fragen Ralf Farke und in allen Angelegenheiten der Verwaltung Claudia Werkmeister.

Wesentliche Erkenntnisse der Dissertation sind im Rahmen des Forschungsprojekts „Integrierte klinische Informationssysteme nach dem Single-Source-Konzept“ in Zusammenarbeit mit dem Institut für Medizinische Informatik entstanden. Für die kollegiale Zusammenarbeit im Zuge dieses Projekts danke ich Philipp Bruland. Der Deutschen Forschungsgemeinschaft danke ich für die Projektförderung unter dem Kennzeichen DU 352/5-1.

Meiner Familie und meinen Freunden danke ich für die Begleitung in dieser intensiven Zeit der Promotion.

Münster, im Mai 2014
Christian Forster

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel motiviert die vorliegende Arbeit. Zuerst wird erläutert, worum es in der betrachteten Anwendungsentwicklung mit XML-Techniken geht. Anhand von Beispielen aus Lehrbüchern und wissenschaftlichen Arbeiten wird ein weit verbreiteter, herkömmlicher Ansatz zur Softwareentwicklung vorgestellt, der sich dadurch auszeichnet, dass für die unterschiedlichen Komponenten der Software jeweils vermeintlich am besten geeignete, aber paradigmatisch unterschiedliche Techniken eingesetzt werden. Die diesem Ansatz inhärenten Paradigmenbrüche und der Umgang damit in Literatur und Praxis wird vorgestellt. Es wird ein Fallbeispiel aus der medizinischen Domäne eingeführt, auf das die Arbeit zur Verdeutlichung von Sachverhalten zurückgreift. Schließlich wird der Untersuchungsgegenstand in der Forschungsfrage verdeutlicht und daraus die Struktur der Arbeit abgeleitet.

1.1 Motivation der XML-basierten Architektur für Webanwendungen

Webanwendungen, also Programme, die ihre Benutzerschnittstelle mit Web-Techniken realisieren und über ein Netzwerk ausgeliefert werden, verzeichnen eine steigende Popularität und immer mehr Anwendungen werden als Webanwendungen implementiert. Eine breite Öffentlichkeit ist durch die Entwicklungen des Web 2.0 [VH07] zu Benutzern von Webanwendungen geworden und auch im betrieblichen Kontext kommt Webanwendungen eine wichtige Bedeutung zu, was die Forschung zu Cloud Computing in der Kategorie Software-as-a-Service (SaaS) behandelt [VHH12]. Aus Sicht des Anwenders bieten Webanwendungen viele Vorteile: Sie müssen clientseitig nicht installiert oder gewartet werden, da sie beim Aufruf der Webadresse geladen werden. Ein Webbrowser ist als Plattform ausreichend, somit ist die Anwendung unabhängig von Betriebssystem und darauf verfügbarer Laufzeitumgebung.

Extensible Markup Language (XML) ist eine Auszeichnungssprache für hierarchisch strukturierte Informationen. Die Sprache ist textbasiert und somit weitgehend interoperabel. XML wird als Lingua Franca des Internets bezeichnet

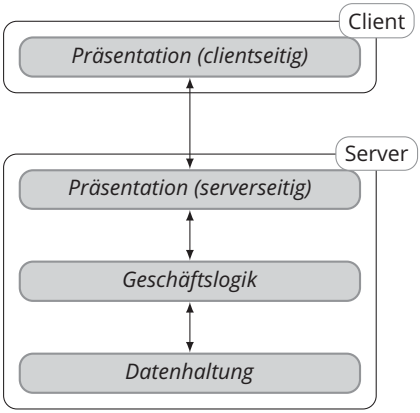


Abbildung 1.1: Gängige Drei-Schichten-Architektur. Dienste der höheren Schicht greifen auf die Dienste, die durch die jeweils niedrigere Schicht bereitgestellt werden, zu; vgl. [LD04].

[Mer03] und ist insbesondere im Bereich serviceorientierter Architekturen [NL05; Jos08] als Kommunikationsformat unverzichtbar.

Die vorliegende Arbeit greift ein konzeptionelles Problem in häufig zu findender Softwarearchitektur auf und zeigt, wie dieses durch den Einsatz von XML und verwandten Techniken vermieden werden kann.

1.2 Paradigmenbruch in der Softwareentwicklung

Software wird häufig in Form einer Schichtenarchitektur entwickelt, bei der Funktionalität nach Dienstmerkmalen gruppiert wird. Diese Dienste stehen in einem hierarchischen Verhältnis, wobei jeder Dienst für den jeweils übergeordneten Dienst Leistungen erbringt und auf den untergeordneten Dienst zugreift [LD04]. Die Aufteilung ist dabei zunächst logischer Art und nicht an Verwendung dedizierter Hardware gebunden. Abbildung 1.1 zeigt das Prinzip anhand eines einfachen Modells, das in den folgenden Kapiteln noch präzisiert wird. Die Schichten Datenhaltung und Geschäftslogik sind serverseitig platziert, die Präsentationsschicht enthält sowohl server- als auch clientseitige Komponenten.

Zur Entwicklung von Webanwendungen kommt häufig eine Drei-Schichten-Architektur zum Einsatz, bei der Datenhaltung, Geschäftslogik und Präsentation in getrennte Schichten separiert werden und unter Verwendung der für die

jeweilige Schicht als am geeignetsten angesehenen Technik entworfen werden. Dass es hierbei zu Brüchen kommen kann, wird u. a. im Lehrbuch der Software-Technik [Bal01] von Balzert deutlich: Demzufolge ist einerseits die Verwendung relationaler Datenbankmanagementsysteme (DBMS) der Normalfall zur Datenspeicherung und das objektorientierte Vorgehen ist der Standard bei Entwicklung von Anwendungssoftware. Andererseits ist die objektorientierte Entwicklung von Anwendungsprogrammen ein Indiz dafür, kein relationales DBMS zu verwenden.

Dies führt zu zunächst inkompatiblen Übergängen zwischen den Schichten, da sowohl die Syntax als auch die zugrunde liegenden Paradigmen unterschiedlich sein können. Bei Webanwendungen, deren Benutzerschnittstelle in Webbrowsern angezeigt wird, kommt noch der Übergang zwischen serverseitiger, meist objektorientierter Technik und clientseitiger Webtechnik hinzu. Werden mittels Webservices externe Dienste eingebunden, so besteht zudem noch die Notwendigkeit, das dabei verwendete, häufig auf XML basierende Nachrichtenformat in die Sprache der Geschäftslogik zu übersetzen.

1.3 Fallbeispiel

Die vorliegende Arbeit basiert auf der Durchführung des Projekts „Integrierte klinische Informationssysteme nach dem Single-Source-Konzept“. Ziel dieses Projektes war die Nutzung von klinischer Behandlungsdokumentation in der klinischen Forschung. Die Dokumentation dieser beiden Domänen erfolgt herkömmlicherweise in getrennten Systemen. Klinische Behandlungsdokumentation liegt im Krankenhausinformationssystem (KIS) vor, während die Dokumentation klinischer Studien separat in studienspezifischen Systemen stattfindet. Somit findet eine redundante Erfassung der Daten statt, die in beiden Domänen benötigt werden. Im Rahmen des Projekts wurde ein Single-Source-Konzept entwickelt, das redundante Datenerfassung eliminiert, indem Daten, die bereits im KIS vorliegen, für Zwecke der Studie wiederverwendet werden [DFB+11].

Um dieses Konzept prototypisch zu implementieren, wurde die Anwendung exchange for trials (x4T) entwickelt [BFD12]. Die Anforderungen an die Software lagen in einer Integration in das bereits genutzte KIS und einer Exportfunktionalität der in einer Studie erhobenen Daten in das dafür von der Standardisierungsorganisation Clinical Data Interchange Standards Consortium (CDISC) spezifizierte Datenformat Operational Data Model (ODM). Die Software besteht aus einem KIS-spezifischen Modul und einem Studienmodul, das als Anwendungsbeispiel dieser Arbeit fungiert.

Das KIS-spezifische Modul kapselt die Interna des jeweiligen KIS und stellt die vom Studienmodul benötigten Funktionen über eine einheitliche Schnittstelle bereit. Dazu wird es für den Datenzugriff auf die Datenbank des KIS angepasst, wozu auch die semantische Anreicherung der Rohdaten mit Termini aus klinischen Begriffssystemen zählt, die zur Identifikation wiederverwendbarer Daten benötigt werden. Außerdem integriert es Einstiegspunkte für das klinische Personal zur Nutzung des Studienmoduls in die routinemäßig verwendete KIS-Softwareumgebung.

Das Studienmodul ist als Webanwendung mittels der in dieser Arbeit vorgestellten XML-basierten Architektur realisiert. Es übernimmt die Datenintegration der Behandlungsdokumentation in die Studiendokumentation auf Grundlage der semantisch annotierten Daten aus dem KIS und bietet eine Benutzeroberfläche für Präsentation und Bearbeitung von Studienformularen. Es bietet ein abgestuftes Rechtemanagement, Datenexportoptionen in verschiedene Formate und einfache deskriptive Auswertungsfunktionen.

1.4 Forschungsfrage und Struktur der Arbeit

Die vorliegende Arbeit stellt einen Ansatz zur Softwarearchitektur vor, der Paradigmenbrüche im Schichtenmodell vermeidet. Dazu wird die weit verbreitete Objektorientierung der Geschäftslogik zugunsten einer XML-basierten Dokumentenorientierung aufgegeben und es werden native XML-Techniken verwendet. Es wird gezeigt, wie sich diese einsetzen lassen, um eine Softwarearchitektur ohne Brüche zu entwickeln.

Zunächst beschäftigt sich Kapitel 2 mit dem Vorgehen zur Softwareentwicklung und konventioneller Software-Architektur anhand der Drei-Schichten-Architektur. Die damit verbundenen Paradigmenbrüche werden beschrieben und es wird erläutert, warum es insbesondere für den objektrelationalen Bruch keine dominante Lösung gibt. Sodann stellt Kapitel 3 verfügbare XML-Techniken vor, die im Rahmen der hier entwickelten XML-basierten Architektur eingesetzt werden. Diese Techniken eignen sich für Datenhaltung (XML Datenbanken), Implementierung der Geschäftslogik (XQuery) und Schnittstellen zu Mensch (XForms) und Maschinen (Webservices). Kapitel 4 stellt vor, wie Software im Hinblick auf diese XML-basierte Architektur entworfen werden kann. Dazu werden die Phasen Analyse, Entwurf, Implementierung und Betrieb herkömmlicher Softwareentwicklung aufgegriffen und unter Berücksichtigung der Erfordernisse der hier vorgestellten XML-basierten Architektur angepasst. Kapitel 5 illustriert die Verwendung des Ansatzes für XML-basierte Webanwendungen an-

hand einer Implementierung. Die darin entwickelten Komponenten behandeln domänenübergreifende Anforderungen, sind zur Wiederverwendung in konkreten Anwendungen gedacht und werden anschließend zur Entwicklung der Anwendung x4T eingesetzt. Wesentliche Erkenntnisse zu der XML-basierten Architektur sind im Rahmen dieser Softwareentwicklung entstanden. Kapitel 6 diskutiert die XML-basierte Architektur und gibt einen Ausblick auf weitere Einsatzszenarien. Kapitel 7 fasst die Kernpunkte der Arbeit zusammen.

2 Grundlagen

In diesem Kapitel wird das grundlegende Fundament der Arbeit gelegt. Es werden Prinzipien der Softwareentwicklung vorgestellt, die sich mit Entwurf und Modellierung von Softwaresystemen befassen. Sodann wird deren Umsetzung anhand konkreter Paradigmen gezeigt und es wird aufgezeigt, wo bei dieser Umsetzung naturgemäß Paradigmenbrüche auftreten und wie sie überbrückt werden. Im Anschluss wird Kritik an dem Ansatz geübt, Paradigmenbrüche im Entwurf in Kauf zu nehmen und dann durch die Einführung von Techniken, die nur der Überbrückung dieses Entwurfsproblems dienen, auszugleichen. Abschließend werden aktuelle Trends der Informatik aufgezeigt, die aufgrund ihrer Beschaffenheit gut in die XML-basierte Softwareentwicklung passen, wie an entsprechenden Stellen der Arbeit gezeigt wird.

2.1 Software-Entwicklung

Softwaretechnik behandelt nach [Bal09] die „[z]ielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen“. Im Rahmen der Wirtschaftsinformatik beschäftigt sich Softwareentwicklung mit Anwendungssystemen [SH05], d.h. Gegenstand der Betrachtung ist die Entwicklung von Software, die einen betrieblichen Anwendungszweck hat und sich so von Systemsoftware oder Unterstützungssoftware unterscheidet.

2.1.1 Ablauf der Entwicklung

Die exponentiell steigende Leistungsfähigkeit und die damit verbundene Komplexitätssteigerung von Computerhardware führte gegen Ende der 1960er Jahre zur Software-Krise [Dij72]. Bis zu diesem Zeitpunkt verursachte die Hardware eines Computers deutlich größere Kosten als die Software und stand im Fokus des Interesses. Die Programmierung dieser oft heterogenen Maschinen wurde als nur lokal interessant angesehen. Da Programme sehr hardwarenah entwickelt wurden, waren sie für die Betreiber anderer Computer nicht von Interesse,

es entstand kein ausgeprägter Austausch und somit auch keine eigene Disziplin der Software-Entwicklung, innerhalb derer sich anerkannte Vorgehensweisen hätten herausbilden können. Dies geschah erst, als die Komplexität der Hardware stieg und gleichzeitig der Preis in die Größenordnung der Software sank. Die bisherigen naiven Ansätze zur Programmierung konnten damit nicht mehr schritthalten und gleichzeitig gab es ein ökonomisches Interesse an Software.

Aus diesen Umständen resultierte der Beginn der systematischen Beschäftigung mit Softwareentwicklung. Seither wurden für die Entwicklung von Anwendungssoftware unterschiedliche Vorgehensmodelle vorgestellt, die im Kern aus den folgenden vier Phasen [SH05] bestehen:

1. Analyse
2. Entwurf (Design)
3. Realisierung
4. Einführung und Nutzung

Bei den Vorgehensmodellen lassen sich klassische und agile Modelle unterscheiden. Klassische Modelle bieten einen stark strukturierenden Rahmen, innerhalb dessen die Phasen bearbeitet werden. Der Umfang der Vorgaben variiert dabei stark. So gibt das Wasserfallmodell [Roy70] lediglich die Inhalte der Phasen und die Übergänge vor, während bspw. das V-Modell XT [35] detaillierte Vorgaben zu nahezu allen in einem Entwicklungsprojekt zu verrichtenden Tätigkeiten enthält.

Die in jüngerer Zeit entstandenen agilen Modelle folgen den Grundlagen des agilen Manifests [11]. Dieses postuliert, dass flexibles Vorgehen mehr zum Projekterfolg beiträgt als das Festhalten an Strukturen und sucht bürokratische Vorgaben möglichst zu vermeiden. Abschnitt 2.5.1 erläutert die Prinzipien genauer.

Unabhängig davon, ob ein klassisches oder agiles Vorgehen eingesetzt wird, werden die Inhalte der genannten Phasen umgesetzt. Das Wasserfallmodell setzt hierbei auf einen sequentiellen Ablauf, der Rückkehr zu Vorphasen und somit Iterationen nur im Fehlerfall erlaubt. Das V-Modell XT unterstützt sowohl einen sequentiellen als auch einen iterativen Ablauf der Entwicklungsphasen und fordert umfangreiche Strukturierung und Dokumentation. Die agilen Methoden zeichnen sich durch Iterationen in tendenziell kurzen Zyklen aus, so dass die einzelnen Phasen häufig durchlaufen werden [RCB10]. Die Abbildung 2.1 zeigt den Ablauf.

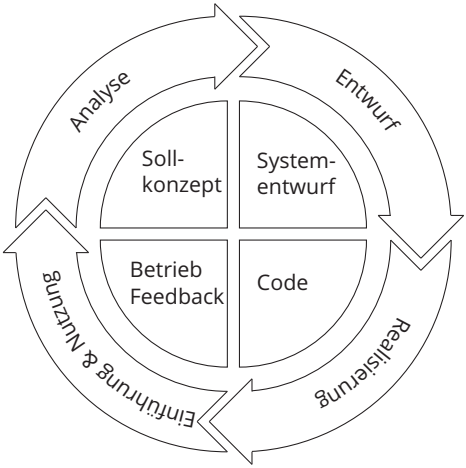


Abbildung 2.1: Ablauf der Phasen in der Softwareentwicklung und deren Resultate. Traditionelle Vorgehensmodelle bearbeiten die einzelnen Phasen umfassend während agile Methoden für kleine Teilprojekte häufig iterieren.

2.1.2 Analyse

Ziel der Analyse ist die Erstellung eines Sollkonzepts, das beschreibt, *was* das zu entwickelnde System leisten muss. Von implementierungsspezifischen Details wird dabei abgesehen und von Technologie wird weitgehend abstrahiert. Die Phase besteht aus der Durchführung einer Ist-Analyse und der Erstellung des Sollkonzepts. Die Ist-Analyse untersucht die existierenden Geschäftsprozesse und organisatorischen Gegebenheiten und legt damit die Grundlage für die Beschreibung des Sollzustandes. Das Sollkonzept enthält funktionale Anforderungen, also solche, die sich aus dem Leistungsumfang ergeben, und nichtfunktionale Anforderungen, die sich auf Realisierung, Projektorganisation und Qualität beziehen. Klassischerweise enthält es zudem eine Betrachtung der zu erwartenden Wirtschaftlichkeit und darf zu diesem Zweck auch technische Anforderungen enthalten, um Realisierungsalternativen vergleichbar zu machen [SH05].

Im Rahmen dieser Arbeit wird die Realisierung durch die XML-basierte Architektur angenommen und es wird an dieser Stelle keine Betrachtung von Alternativen vorgenommen. Daher wird das Sollkonzept hier als reines Fachkonzept

aufgefasst. Eine Diskussion der XML-basierten Architektur erfolgt in Kapitel 6.

Bei Verwendung eines sequentiellen Modells wird das Sollkonzept in einem umfassenden Dokument dargestellt, bei agilen Methoden wird lediglich das Konzept der Funktionen erstellt, die innerhalb der zugehörigen Iteration entstehen sollen.

2.1.3 Entwurf

Der Entwurf beschreibt, *wie* das System die während der Analyse definierten Leistungen erbringt und bereitet die Implementierung unmittelbar vor. Innerhalb der Phase wird ausgehend vom Sollkonzept durch Detaillierung und Festlegung von Implementierungsaspekten das Softwaredesign bestimmt. Zunächst wird ein Architekturentwurf entwickelt, der iterativ ausgebaut wird. Die Architektur bestimmt das Gerüst des Systems, indem sie Aufgaben in Architekturbausteine gliedert und einen Integrationsrahmen für die Kommunikation der Bausteine darstellt [PBG07]. Bei hoher Komplexität einzelner Bausteine können diese wiederum durch einen Architekturentwurf strukturiert werden. Somit werden die frühesten und grundlegendsten Designentscheidungen durch die Architektur festgelegt. Dazu gehört auch die Wahl der Programmierparadigmen, also der grundlegenden Modellierungs- und Umsetzungsstile. Je nach Paradigma stehen für die Modellierung der Bausteine unterschiedliche Methoden zur Verfügung, beispielsweise die Diagrammarten der Unified Modeling Language (UML) für objektorientierte Entwicklung [HKKR05] oder das Entity-Relationship-Modell (ERM) für relationale Datenmodelle [Che76]. XML-Erweiterungen für UML finden sich in [CSF00; LLY06], eine Erweiterung für ERM in [LSR03] und eine Übersicht weiterer XML-Modellierungstechniken in [CHL11].

Auf den Architekturentwurf folgend wird das Design der Bausteine erstellt und ggf. iterativ präzisiert, bis ein implementierungsreifer Entwurf existiert. Hierbei werden auch die Anforderungen an diese Bausteine, abgeleitet von den analysierten Anforderungen an das gesamte System, spezifiziert.

In den klassischen Methoden wird der Architekturentwurf durch spezialisierte Softwarearchitekten durchgeführt und vor Beginn der Implementierung möglichst vollständig spezifiziert. Bei agilem Vorgehen mit wenig stark ausgeprägter Rollenspezialisierung der Entwickler entsteht die Architektur kollaborativ und nur in dem für die Implementierung der kommenden Iteration benötigten Umfang.

2.1.4 Realisierung

Während der Realisierung entsteht der Code für das zuvor spezifizierte System. Dies ist, abgesehen von den Ansätzen zur modellgetriebenen Softwareentwicklung [MSUW02], kein streng formaler Transformationsprozess, aber für die Umsetzung häufig auftretender Strukturen existieren Entwurfsmuster aus erprobten Umsetzungsstrategien. Für die objektorientierte Entwicklung haben sich die in [GHJV94] beschriebenen Entwurfsmuster etabliert. Für funktionale Sprachen gibt es dazu analoge Vorschläge[Gib06; CBK10].

Die Realisierung geschieht durch Programmieren und Testen und wird durch Softwareentwicklungswerkzeuge in unterschiedlichem Umfang unterstützt. Einige sehr einfache Werkzeuge unterstützen Teilaspekte, während umfangreiche Werkzeuge bereits die Analyse- und die Entwurfsphase unterstützen und aus den Modellen Programmcode und Testfälle oder Teile davon generieren [SV05]. Der geeignete Grad der Werkzeugunterstützung hängt von vielen Parametern wie der Komplexität des Systems, den für die gewählte Programmiersprache verfügbaren Tools oder der Erfahrung der Entwickler ab.

Tests dienen der Erzielung einer hohen Softwarequalität und werden als integraler Bestandteil der Softwareentwicklung angesehen. Qualitätsmaße sind einerseits die Übereinstimmung des Systems mit der Spezifikation, was als Verifikation bezeichnet wird, und die Übereinstimmung des Systems mit den Anforderungen der Anwender, was als Validierung bezeichnet wird. Bis auf die sehr aufwändige und nur für einfache Programme nutzbare formale Beweisführung [Dij68; Hoa69] sind Tests nur geeignet, Fehler aufzudecken, aber nicht die Korrektheit eines Programms zu beweisen [Gol11]. Da in praxi nicht alle möglichen Zustände der Software getestet werden können, zielt die Erstellung der Testfälle darauf ab, diejenigen, die mit der höchsten Wahrscheinlichkeit Fehler entdecken, auszuwählen [MSBT04].

Während der Entwicklung steht jedem Schritt der Implementierung ein entsprechender Test gegenüber. Modultests testen als feingranulare Einheit einzelne, abgrenzbare Bereiche der Software in isolierter Umgebung. Integrationstests prüfen die Funktionsfähigkeit von aus Modulen zusammengesetzten Subsystemen. Systemtests testen das vollständige System im Hinblick auf Funktionalität des Gesamtsystems. Die o. a. Tests dienen der Verifikation. Abnahmetests finden als Validierung nach der Installation in der Produktionsumgebung statt [Gol11]. Durch diese Abstufung der Tests sollen Fehler möglichst früh erkannt werden, da die Kosten für die Fehlerbehebung geringer sind, je früher der Defekt erkannt wird [PY09]. Neben diesen, die funktionalen Anforderungen testenden Verfahren, stellen Tests auf Performanz eine wichtige Testkategorie dar, um die

Leistungsfähigkeit des Systems sicherzustellen [Mol09].

Je nach verwendeter Testmethode lassen sich Tests als statisch und dynamisch klassifizieren. Statische Testmethoden untersuchen den Quellcode des Systems, ohne ihn auszuführen. Dazu zählen die Analyse von Kontrollstruktur, Datenfluss, Komplexität, Schnittstellen und Kommentaren sowie manuell ausgeführte Reviews, die Suche nach doppeltem Code und Überprüfung der Einhaltung guter Programmierpraxis [Gol11]. Während die statischen Methoden immer *White-Box-Verfahren* sind, bei denen Kenntnisse aus der inneren Struktur in den Testfällen verwendet werden, können dynamische Methoden sowohl White- als auch *Black-Box-Verfahren* sein, bei denen ohne Kenntnis der inneren Struktur das Verhalten anhand der vorgesehenen Schnittstellen getestet wird. Unter allen Testmethoden gilt das dynamische funktionsorientierte Testen als die am meisten verwendete Methode [Gol11]. Dabei werden Testfälle aus der Spezifikation abgeleitet und es wird geprüft, ob das System korrekt reagiert. Da nicht alle zulässigen Eingabewerte getestet werden können, teilt man Eingabeparameter in Äquivalenzklassen ein, von denen man annimmt, dass sie ein gleichartiges Verhalten bewirken, aus denen man repräsentative Werte testet. Zudem werden im Rahmen einer Grenzwertanalyse Parameter, die an der Grenze des zulässigen bzw. unzulässigen Wertebereichs liegen, getestet. Weitere dynamische Methoden sind diversifizierende Testmethoden, wie der *Back-to-Back-Test*, bei denen unterschiedliche Versionen der Software mit denselben Eingabedaten getestet werden und die Ergebnisse verglichen werden, und statistische Tests bei denen Eingabedaten zufällig erzeugt werden und mit manuell bestimmten Sollwerten verglichen werden [Gol11].

Die klassischen Vorgehensweisen der Softwareentwicklung führen Programmieren und Testen tendenziell als stärker separierte Schritte an. Häufig findet man die Empfehlung, diese auf verschiedene Entwickler zu verteilen, während agile Methoden das Testen stärker integrieren. Dies kann bis zur testgetriebenen Entwicklung reichen, bei der die Testfälle vor dem Programmcode erzeugt werden und der Programmcode dann so geschrieben wird, dass die Tests bestanden werden.

2.1.5 Einführung und Nutzung

Als Ergebnis dieser Phase wird die zuvor entwickelte Software in Betrieb genommen. Dies kann je nach Anforderungen zu einem Stichtag oder sukzessiv geschehen. Die technische Einführung wird durch einen Umstellungsplan begleitet. Darin wird festgelegt, wie gegebenenfalls Altdaten migriert werden, Benutzer für das neue System geschult werden und welche Dokumentation zu

hinterlegen ist.

Die klassischen Vorgehensweisen erfordern eine komplette Fertigstellung der Software vor der Einführung, die agilen Methoden erlauben, auch die Ergebnisse einzelner Iterationen in Betrieb zu nehmen, da diese jeweils vollständig getestete Funktionen enthalten.

Nach der Inbetriebnahme eines Softwaresystems ist die Entwicklung meist nicht abgeschlossen. Da durch das Testen keine Fehlerfreiheit erreicht werden kann, kann es sein, dass Fehler erst im Betrieb zufällig oder durch Evaluation der Anwender entdeckt werden. Ebenso kann es sein, dass sich die Spezifikation erst im Betrieb als fehlerhaft herausstellt, so dass die Software zwar die Spezifikation erfüllt, aber nicht den Erwartungen der Anwender entspricht, bspw. weil sie zu langsam oder nicht benutzerfreundlich genug ist. Außerdem sind Anforderungen an eine Software nicht statisch, sondern Änderungen im Zeitablauf unterworfen, so dass Anpassungen an neue Gegebenheiten nötig werden können, bspw. weil sich externe Schnittstellen ändern oder neue gesetzliche Anforderungen gelten. Die Wartung von Software ist demnach korrigierend, perfektionierend oder adaptiv [LS80]. Andere Autoren unterscheiden nur nach Art der Modifikation Fehlerbehebungen und Verbesserungen [KTM+99].

Ergibt sich während des Betriebs des Systems die Notwendigkeit zur Anpassung, so beginnt der Entwicklungszyklus erneut, so dass man von Software-Evolution spricht [MD10].

2.2 Traditionelle Paradigmen und Techniken

Für die Systemmodellierung des Softwareentwurfs, die die Anforderungen der Analysephase in ein implementierbares Modell des Systems transformiert, existieren unterschiedliche Sichtweisen darauf, wie betriebswirtschaftliche Objekte, Funktionen und Organisationsstrukturen abgebildet werden. Bei Anwendungssystemen findet sich häufig eine Schichtenarchitektur, die Systemfunktionen nach Aufgaben in Schichten einteilt [JPMM04; ACKM04]. Bei der häufig eingesetzten Drei-Schichten-Architektur werden bspw. Geschäftslogik, Datenhaltung und Präsentationsschicht unterschieden und mit jeweils passenden Techniken implementiert.

2.2.1 Objektorientierung in der Geschäftslogik

Seit den 1990er Jahren hat sich die objektorientierte Betrachtungsweise zur Modellierung von Sachverhalten, die in Informationssystemen abzubilden sind, um-

fangreich durchgesetzt. Damit geht die Entwicklung und Verwendung dazu passender, die Objektorientierung unterstützender Programmiersprachen einher.

Die Objektorientierung [RBP+91] versteht Software als System von identifizierbaren Objekten, die einen Zustand und Verhaltensmöglichkeiten haben. Das Basiskonstrukt dazu ist die Klasse, die Objekte derselben Art beschreibt. Durch Instanziierung einer Klasse werden konkrete Objekte erzeugt. Die Klasse gibt an, über welche Attribute, deren Werte den Zustand bestimmen, und Methoden, die das Verhalten ermöglichen, ihre Objekte verfügen.

Weiterhin wesentliche Prinzipien für die objektorientierte Sichtweise sind Vererbung, Kapselung, Nachrichtenaustausch und Polymorphie [Arm06].

Vererbung ermöglicht es, dass Klassen die Eigenschaften und das Verhalten anderer Klassen, von denen sie erben, übernehmen. Die erbende Klasse ist also eine Spezialisierung, die nur zusätzliche Eigenschaften und Verhaltensweisen selbst bereitstellt. Hierdurch lassen sich unterschiedliche Klassen mit gemeinsamen Eigenschaften und Methoden herstellen, ohne dass diese redundant angelegt werden müssen. Das Ersetzbarkeitsprinzip [LW92] besagt, dass Objekte stärker spezialisierter Klassen immer dann verwenden werden können, wenn Objekte generellerer Klassen erwartet werden.

Das Prinzip der Kapselung besagt, dass Objekte ihre Zustandsinformationen vor der Umgebung verbergen und den Zugriff darauf nur indirekt über durch von ihnen angebotene Methoden zulassen. Hierdurch ist sichergestellt, dass die ausgeführten Methoden zu den Zustandsdaten passen und keine inkonsistenten Manipulationsoperationen stattfinden.

Nachrichtenaustausch ist der Vorgang des Methodenaufrufs und der damit verbundenen Informationsübertragung. Durch den Nachrichtenaustausch werden Objekte darüber benachrichtigt, dass sie eine Methode, die mit Namen und Parametern als Signatur identifiziert ist, ausführen sollen. Welche Implementierung dabei zum Einsatz kommt, ist dem Objekt überlassen.

Der Nachrichtenaustausch ist die Voraussetzung für Polymorphie. Sie bezeichnet die Fähigkeit von Objekten, die zu Klassen einer Vererbungshierarchie gehören, unterschiedlich auf identische Methodenaufrufe zu reagieren. Eine generelle Klasse kann also eine Methode bereitstellen, die von ihr und den erbenden Klassen genutzt wird. Falls aber eine erbende Klasse dieses Verhalten anders implementieren soll, kann sie eine Methode mit identischer Signatur bereitstellen.

Zur Verdeutlichung diene folgendes Beispiel zur Verwaltung von Daten in klinischen Studien. Abbildung 2.2 zeigt das zugehörige Klassendiagramm in der UML-Notation.

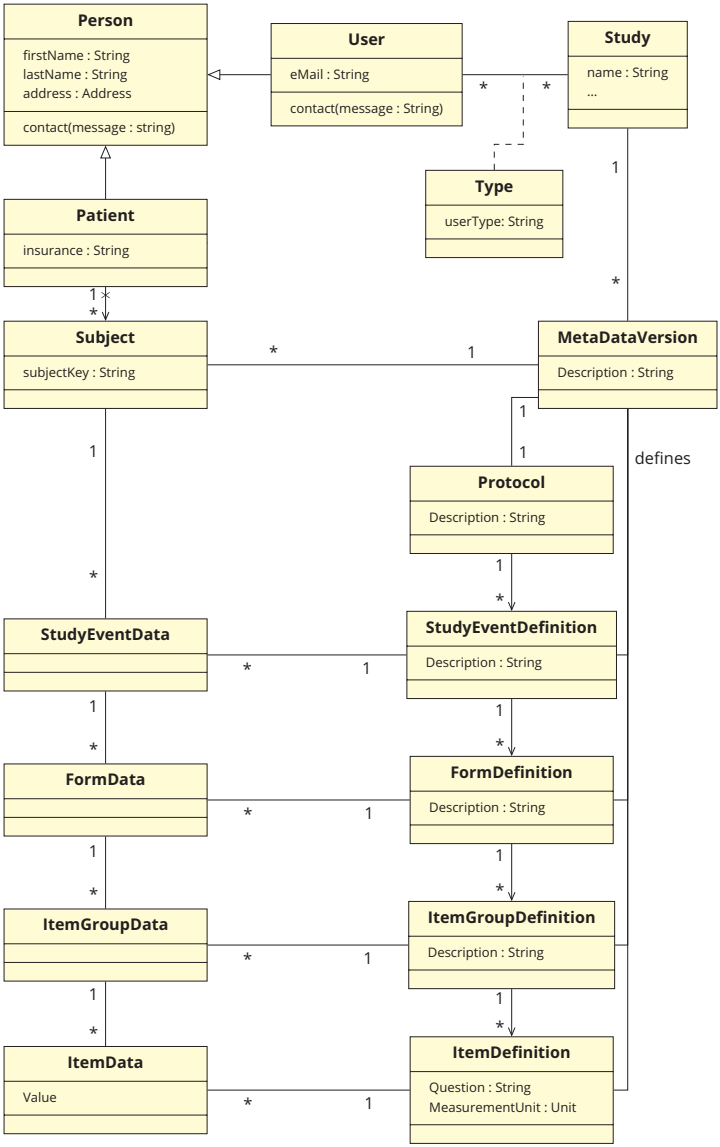


Abbildung 2.2: Objektorientierte Darstellung eines einfachen Studienmanagementsystems, die sich an das Operational Data Model (ODM) anlehnt.

Alle Personen werden als Klasse Person modelliert. Sie verfügen über einen Vor- und Nachnamen sowie eine hier nicht näher spezifizierte Adresse und können durch Nachrichten kontaktiert werden. Mitarbeiter werden in der Klasse User modelliert, die von der Klasse Person erbt. Als Mitarbeiter erhalten Personen zudem noch eine E-Mail-Adresse und lassen sich einer oder mehreren Studien zuordnen, wobei diese Assoziation einen Typ, der ihre Tätigkeit im Rahmen der Studie beschreibt, aufweist. Die Methode contact() der Oberklasse Person wird in der Klasse User überschrieben. Hierbei wird von Polymorphie Gebrauch gemacht. Wenn ein Objekt der Klasse Person die Nachricht erhält, dass die Methode contact() ausgeführt werden soll, unterscheidet sich das Verhalten danach, ob es eine direkte Instanz der Klasse Person oder Patient oder eine Instanz der Klasse User ist. So könnte die Methode contact() bei ersteren z. B. veranlassen, dass ein Brief an die bekannte Adresse geschickt wird, während bei letzteren eine E-Mail versendet wird.

Die Dokumentation einer Studie, die hier angelehnt an den Standard Clinical Data Interchange Standards Consortium (CDISC) Operational Data Model (ODM) modelliert wird, besteht aus hierarchisch geschachtelten Elementen. Diese Elemente werden für eine Version der Studie im Element MetaDataVersion definiert und können in diesem Rahmen mehrfach verwendet werden. Innerhalb dieses Elements existieren die Elemente StudyEventDefinition, die Studienereignisse definieren, FormDefinition, die die Dokumentationsformulare für ein Studienereignis enthalten, ItemGroupDefinition, die zusammengehörige Teile eines Dokumentationsformulars bündeln und ItemDefinition, die die zu erhebenden Merkmale definieren. Übergeordnete Elemente enthalten Verweise auf die in ihnen vorkommenden untergeordneten Elemente. Die tatsächliche Anwendung der Dokumentationselemente wird durch das Element Protocol bestimmt, das einmal je MetaDataVersion existiert und auf die zur Dokumentation verwendeten Elemente StudyEventDefinition verweist. Von diesen ausgehend wird durch die oben erläuterte hierarchische Beziehung die Dokumentationsstruktur bis auf die Ebene der ItemDefinition aufgebaut.

Wenn ein Patient zu einem Studienteilnehmer einer bestimmten Studie wird, so erhält er ein Pseudonym, da die im Rahmen der Studie erhobenen Daten keine Identifizierbarkeit der echten Person ermöglichen dürfen. Die Navigationsrichtung der Assoziation zwischen der Klasse Patient und der Klasse Subject, die einen Studienteilnehmer darstellt, ist daher nur vom Patienten zum Studienteilnehmer möglich. Die Struktur der erhobenen Daten spiegelt durch die Elemente StudyEventData, FormData, ItemGroupData und ItemData die Hierarchie der Definitionselemente wider. Jedes dieser Elemente enthält einen Verweis auf das

jeweils definierende Element.

Die Objektorientierung bietet hinsichtlich der strukturierten Entwicklung, die sie weitgehend abgelöst hat, einige Vorteile [SH05]. Insbesondere vorteilhaft sind folgende Eigenschaften:

- Daten und Funktionen werden in einem einheitlichen Modell entwickelt und sind somit konsistent.
- Die Fokussierung auf Objekte passt zu der geschäftsprozessorientierten[1] Sichtweise auf Unternehmen.
- Die Wiederverwendbarkeit von Komponenten wird durch die Eigenschaften der Objektorientierung gefördert.

Die objektorientierte Programmierung wird durch eine Reihe Programmiersprachen unterstützt, darunter populäre wie Java, C++, Python, Objective-C und Ruby.

2.2.2 Relationales Datenmodell in der Persistenzschicht

Zur Speicherung strukturierter betrieblicher Daten haben sich DBMS, denen das relationale Datenmodell [Cod70] zu Grunde liegt, weitgehend durchgesetzt und sind in der Literatur umfangreich beschrieben, bspw. in [Vos08] und [EN02]. Das relationale Datenmodell fasst Datensätze als Tupel von Attributen auf, die zeilenweise in Tabellenform gespeichert werden. Die Identität eines Tupels ist dabei durch die Werte seiner Attribute bestimmt, so dass identische Tupel in einer Tabelle, sofern das DBMS sie überhaupt erlaubt, nicht unterscheidbar und somit meist sinnlos sind. Das Datenbankschema gibt die Form der Tabellen, die Domänen der Attribute, die Beziehungen der Datensätze untereinander und Integritätsbedingungen an. Attribute, die ein Tupel eindeutig identifizieren, heißen Schlüsselattribute. Der Primärschlüssel ist das zur Identifizierung verwendete Schlüsselattribut. Einem Primärschlüssel kommt eine besondere Bedeutung zu, da er für den Zugriff auf das Tupel und die Sicherstellung der Integrität benötigt werden. Verweist ein Attribut auf einen Primärschlüssel, so heißt es Fremdschlüssel. Eine wichtige Form der Integritätsbedingung ist die Inklusionsabhängigkeit. Sie beschreibt die Abhängigkeit eines Tupels in einer referenzierenden Tabelle von der Existenz eines Tupels in einer referenzierten Tabelle. Die Werte

[1] Hier wird der Prozessbegriff nach [BS04] zu Grunde gelegt, gemäß dem ein Prozess eine inhaltlich abgeschlossene, zeitliche und sachlogische Abfolge von Funktionen ist, die zur Bearbeitung eines betriebswirtschaftlich relevanten Objektes notwendig sind.

der Attribute der referenzierenden Tabelle, für die die Inklusionsabhängigkeit gilt, müssen mit den Werten der entsprechenden Attribute der referenzierten Tabelle übereinstimmen. Mit Inklusionsabhängigkeiten lassen sich referentielle Integrität und Spezialisierungs- bzw. Generalisierungsbeziehungen ausdrücken.

Schemaentwurf Eine wichtige Rolle für den Entwurf eines Datenbankschemas spielt der Begriff der *Normalisierung* [Cod71] zit. n. [Vos08]. Ziel der Normalisierung ist es, die redundante Speicherung von Werten zu vermeiden, um die Datenbank vor inkonsistenten Zuständen zu schützen. In Datenbanken, die nicht normalisiert sind, können solche Anomalien bei Einfüge-, Lösch- und Änderungs-Operationen auftreten. Die Normalisierung relationaler Schemata wird ausführlich in der Literatur behandelt, bspw. in [Vos08; EN02]. Die Grundzüge werden im Folgenden für die erste bis dritte Normalform sowie die Boyce-Codd-Normalform (BCNF) zusammengefasst, die jeweils aufeinander aufbauen. Wichtig ist dabei das Konzept der *funktionalen Abhängigkeit*, mit dem bestimmte Restriktionen bezüglich zulässiger Tupel ausgedrückt werden. Die funktionale Abhängigkeit $X \rightarrow Y$ bedeutet, dass die möglichen Tupel einer Attributmenge Y der Relation durch die Tupel einer Attributmenge X der Relation bestimmt werden.

Ob ein Schema eine bestimmte Normalform einhält, lässt sich mit folgenden Tests überprüfen:

- *erste Normalform (1NF)*: „Die Relation sollte keine nicht atomaren Attribute oder verschachtelten Relationen enthalten." [EN02]
- *zweite Normalform (2NF)*: „In Relationen, deren Primärschlüssel mehrere Attribute enthalten, sollte kein Nichtschlüsselattribut funktional von einem Teil des Primärschlüssels abhängen." [EN02]
- *dritte Normalform (3NF)*: „Eine Relation sollte kein Nichtschlüsselattribut enthalten, das funktional von einem anderen Nichtschlüsselattribut (oder einer Menge von Nichtschlüsselattributen) bestimmt wird. Das heißt, es sollte keine transitive Abhängigkeit eines Nichtschlüsselattributs vom Primärschlüssel bestehen." [EN02]
- *BCNF*: „Ein Relationsschema R ist in BCNF, falls für jede nicht triviale funktionale Abhängigkeit $X \rightarrow A\,X$ ein Superschlüssel von R ist." [EN02] Dies stellt eine Verschärfung gegenüber der 3NF insofern dar, dass eine Attributmenge nur dann funktional abhängig sein darf, wenn sie von einer Attributmenge abhängt, die nur ein einziges Tupel identifizieren kann.

Damit ein Schema die entsprechende Normalform einhält, muss es geprüft werden und ggf. so transformiert werden, dass die verletzten Bedingungen eingehalten werden:

- *1NF*: „Erstelle für jedes nicht atomare Attribut oder jede verschachtelte Relation neue Relationen." [EN02]
- *2NF*: „Zerlege die Relation und erstelle eine neue für jeden partiellen Schlüssel mit seinen abhängigen Attributen. Erhalte die (restliche) Relation mit dem ursprünglichen Primärschlüssel und Attributen, die von diesem funktional voll abhängig sind." [EN02]
- *3NF*: „Zerlege die Relation und erstelle eine neue, die das bzw. die Nichtschlüsselattribute beinhaltet, die funktional von anderen Nichtschlüsselattributen bestimmt werden." [EN02]

Die Zerlegung einer Relation, die in 3NF, aber nicht BCNF ist, ist nicht in jedem Fall so möglich, dass sämtliche Informationen und funktionale Abhängigkeiten erhalten bleiben [Maj92].

SQL Die von der International Organization for Standardization (ISO) standardisierte Sprache für die Interaktion mit relationalen Datenbankmanagementsystemen heißt Structured Query Language (SQL) und liegt derzeit in der Version SQL:2011 vor. Aus logischer Sicht lassen sich die Bestandteile der Sprache in Elemente zur Datendefinition (Data Definition Language, DDL), Datenmanipulation (Data Manipulation Language, DML) sowie Datenverwaltung (Data Administration Language) unterscheiden. Mit den Kommandos der DDL werden das Datenbankschema und seine Integritätsbedingungen erzeugt, mit der DML werden Lese- und Schreibzugriffe durchgeführt und mit der DAL werden Einstellungen der Datenbank, wie bspw. Zugriffsberechtigungen, vorgenommen.

Die Unterstützung der DBMS-Hersteller für den Standard ist unterschiedlich ausgeprägt [10]. Eine umfangreiche Darstellung zu SQL:1999 findet sich in [MS02]. Die in neueren Versionen hinzugekommenen Eigenschaften werden in [EM04; EMK+04; Zem12] erläutert.

Mit der Einführung von strukturierten Datentypen und typisierten Tabellen in SQL:1999 wurde SQL um ein Konzept aus Objektorientierung zum objektrelationalen Modell erweitert [Mel03]. Strukturierte Datentypen können vom Nutzer des DBMS definiert und verwendet werden. Die Attribute strukturierter Datentypen können wiederum aus strukturierten Datentypen oder atomaren

Typen bestehen. Außerdem kann ein strukturierter Datentyp einen anderen erweitern, so dass eine Hierarchie mit Vererbung der Eigenschaften entsteht. Typisierte Tabellen basieren auf strukturierten Datentypen und eignen sich zur Speicherung von Instanzen. Durch das Einfügen eines Tupels in eine typisierte Tabelle wird aus dem Tupel ein Objekt mit eigener, von seinen Werten unabhängiger Identität im Sinne der Objektorientierung.

Die Umsetzung der objektorientierten Bestandteile des SQL-Standards ist mit spezifischen Performanzproblemen behaftet [Mur99]. Bei Verwendung eines im Kern relationalen DBMS und Implementierung der objektorientierten Erweiterung ist der Hersteller mit dem in Abschnitt 2.3.1 erläuterten Paradigmenbruch konfrontiert, der noch nicht allgemein zufriedenstellend überwunden ist. Ein breiter praktischer Einsatz der objektorientierten Eigenschaften von SQL ist jedenfalls derzeit nicht zu beobachten.

Eine Erweiterung der Version SQL:2003 umfasst die Unterstützung von XML in SQL. Diese enthält das Mapping von XML-Dokumenten in das relationale Modell, die Unterstützung von XML-Datentypen, das Publizieren von relationalen Daten als XML-Dokumente und die Ausführung von XQuery- und XPath-Ausdrücken. Bei Verwendung eines sowohl aus relationalen Schemata als auch XML-Dokumenten bestehenden Datenmodells mag dies eine vorteilhafte Ergänzung sein, allerdings bleibt für den Teil, der das relationale Datenmodell mit objektorientierter Geschäftslogik verwendet, der in Abschnitt 2.3.1 aufgezeigte Paradigmenbruch bestehen.

Modellierung Für die konzeptionelle Modellierung relationaler Datenmodelle hat sich ERM etabliert. Abbildung 2.3 zeigt ein ERM-Diagramm für die relationale Datenmodellierung der aus Abschnitt 2.2.1 bekannten Benutzerverwaltung. Aspekte des Verhaltens, wie die Methoden in der objektorientierten Sichtweise, werden dabei konzeptbedingt nicht modelliert. Im Allgemeinen werden bei dieser Umwandlung Klassen zu Entity-Typen und Eigenschaften der Objekte zu Attributen. Assoziationen zwischen Klassen werden zu Relationship-Typen. So enthält die Darstellung die durch Rechtecke gekennzeichneten Entity-Typen Person, User, Address und Study. Deren Attribute werden in Ellipsen an die jeweiligen Entity-Typen angehängt. Schlüsselattribute, die die Identität einer Entität kennzeichnen, werden unterstrichen. Relationship-Typen werden durch Rauten dargestellt, die Kardinalität wird an den verbindenden Kanten vermerkt.

Es ist anzumerken, dass im ursprünglichen relationalen Modell keine Vererbung vorgesehen ist und diese auch in der Originalfassung des ERM [Che76] nicht zu finden ist. Konzepte zu Aggregation und Generalisierung finden sich

daher auch erst in aufbauenden Arbeiten, z. B. in [SS77]. In der ERM-Darstellung wird die Notation mittels eines Dreiecks genutzt, um Spezialisierungen anzuzeigen. Die Spitze des Dreiecks zeigt dabei in Richtung der Generalisierung. Die Beschriftung gibt an, ob die Zerlegung partiell (p) oder total (t) sowie disjunkt (d) oder nicht-disjunkt (n) ist. Eine partielle Zerlegung bedeutet, dass Entities sowohl von dem generellen als auch von einem speziellen Entity-Typ sein können, während bei totaler Zerlegung Entities von einem speziellen Entity-Typ sein müssen. Bei disjunkter Zerlegung können Entities nur zu einem speziellen Entity-Typ gehören, bei nicht-disjunkter Zerlegung ist die Zugehörigkeit zu mehreren Typen möglich. Im Beispiel wird die Vererbungsbeziehung zwischen Person und User bzw. Patient zu einer *IS-A*-Beziehung. Da Personen erzeugt werden sollen, die keine Spezialisierung aufweisen, wird die Beziehung als partiell modelliert. Es ist nicht vorgesehen, dass eine Person mehrere dieser Rollen annehmen kann, daher wird die Beziehung zudem disjunkt modelliert. Der Relationship-Typ P_A ordnet Person eine Adresse zu. Hier wird angenommen, dass eine Person nur eine Adresse hat, unter einer Adresse aber mehrere Personen zu finden sein können, daher ist die Kardinalität 1:n. Die Zuordnung von User und Study ist eine m:n-Beziehung, die durch den Relationship-Typ U_S realisiert ist und im Attribut Type die Rolle des Mitarbeiters im Rahmen der Studie angibt. Ein User kann also mehreren Studien zugeordnet sein und eine Studie kann mehrere User aufweisen.

Für die Umsetzung des ERM-Diagramms in das Relationenmodell gelten folgende Regeln [Vos08]:

1. „Jeder Entity-Typ wird in ein Relationenschema transformiert."

2. „Jeder Relationship-Typ wird ebenfalls in ein Relationenschema transformiert, es sei denn, es handelt sich um eine zweistellige 1:1- oder 1:n-Beziehung; in diesen Fällen reicht die Hinzunahme von Attributen zu bereits existierenden Relationenschemata."

3. „IS-A-Beziehungen werden allein über INDs [Inklusionsabhängigkeiten] ausgedrückt."

Um das in Tabelle 2.1 abgebildete Relationenmodell zu erhalten, werden die genannten Regeln folgend angewendet. Zunächst werden die vier Entity-Typen gemäß Regel 1 in Relationenschemata transformiert. Laut Regel 2 wird der Relationship-Typ P_A mit der Kardinalität 1:n durch ein weiteres Attribut in einer bestehenden Relation ausgedrückt. In diesem Fall wird das Schlüsselattribut der Adresse als Fremdschlüssel in das Relationenschema Person aufgenommen. Der

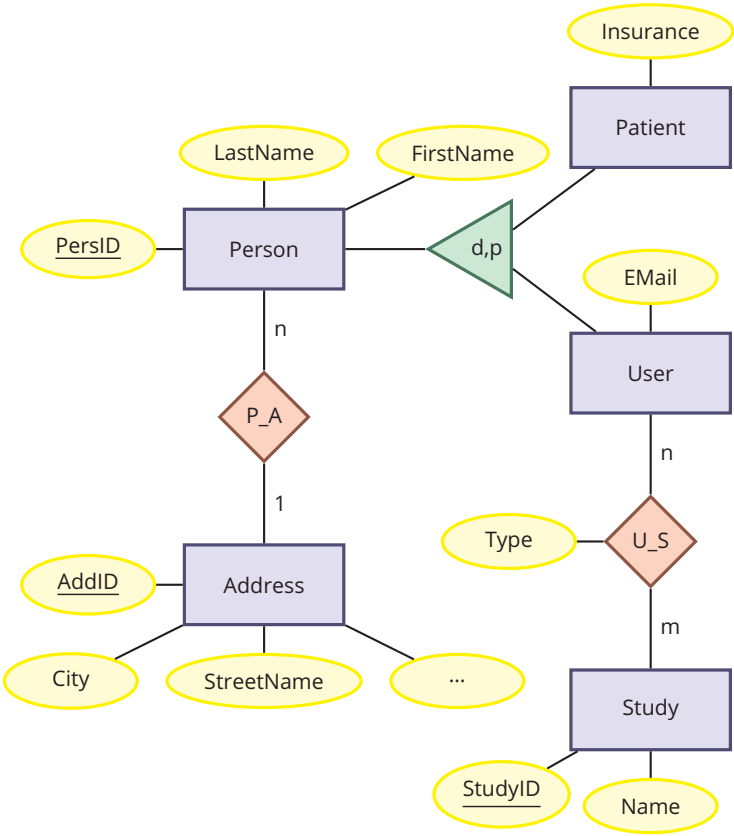


Abbildung 2.3: ERM-Diagramm der Datenhaltung einer Benutzerverwaltung.

Relationship-Typ U_S muss aufgrund seiner m:n-Kardinalität als eigenes Relationenschema modelliert werden, dessen Schlüssel aus den Schlüsselattributen der Relationenschemata Person und Study zusammengesetzt ist.

Die Spezialisierungsbeziehung zwischen Person und User wird gemäß Regel 3 abgebildet. Das Relationenschema User verwendet im Rahmen einer schlüsselbasierten Inklusionsabhängigkeit dasselbe Schlüsselattribut wie Person.

Einsatz Relationale DBMS haben seit ihrer Verbreitung Anfang der 1980er Jahre einen hohen Reifegrad erreicht. Sie unterstützen die Bearbeitung von Operation durch Transaktionen nach dem ACID-Prinzip. Dieses Akronym steht für die folgenden Eigenschaften, die die grundlegenden Zusicherungen für ein verlässliches DBMS darstellen:

- Atomicity: Atomarität bedeutet, dass eine Transaktion atomar ausgeführt wird, d. h. vollständig oder überhaupt nicht. Zustände, die während der Transaktion auftreten, sind von außen nicht zugreifbar. Kommt es während der Transaktionsverarbeitung zu einem Fehler, wird der vorherige Zustand wieder hergestellt.

- Consistency: Die Zusicherung der Konsistenz bewirkt, dass eine Datenbank, die vor Beginn der Transaktion in einem konsistenten Zustand war, auch nach dieser wieder in einem konsistenten Zustand ist. Würde durch die Transaktion eine Integritätsbedingung verletzt, wird dies als Fehler gewertet und die Transaktion abgebrochen.

- Isolation: Das Prinzip der Isolation bedeutet, dass mehrere Transaktionen, die gleichzeitig ablaufen, unabhängig voneinander ausgeführt werden. Das DBMS stellt sicher, dass Transaktionen, die auf dieselben Daten zugreifen, sich nicht gegenseitig beeinflussen, sondern jeweils einen konsistenten Datenbankzustand vorfinden.

- Durability: Die Dauerhaftigkeit sorgt dafür, dass Daten, die durch eine Transaktion verändert wurden, nach Abschluss der Transaktion gespeichert sind und bei einem Systemausfall nach dem Hochfahren wieder verfügbar sind.

Üblicherweise verfügen DBMS über eine Benutzer- und Rechteverwaltung, sind für hohe Transaktionsraten optimiert und bieten neben den durch die ISO standardisierten SQL-Befehlen noch herstellerspezifische Erweiterungen.

Mit diesen und weiteren Eigenschaften haben sich relationale DBMS zum defacto Standard für betriebliche Anwendungen etabliert.

Tabelle 2.1: Relationenschemata und Daten des in Abbildung 2.3 gezeigten ERM-Diagramms. Die Spezialisierung von Person zu User ist durch eine Inklusionsabhängigkeit realisiert. Der Primärschlüssel der Relation User ist das Attribute PersID, das in der Relation Person ebenfalls enthalten sein muss.

Person

PersID	LastName	FirstName	Address.ID
forster	Forster	Christian	1
mustermann	Mustermann	Max	2
musterfrau	Musterfrau	Klara	3
doe	Doe	John	3
roe	Roe	Jane	4

Address

AddID	City	StreetName	...
1	Münster	Leonardo-Campus 3	
2	Münster	Albert-Schweitzer-Campus 1	
3	Münster	Albert-Schweitzer-Campus 3	
4	Köln	Heidestr. 17	

User

PersID	EMail
forster	christian.forster@ercis.de
mustermann	max@example.org
musterfrau	klara.musterfrau@example.org

Patient

PersID	Insurance
doe	KK1
roe	KK2

Study

StudyID	Name
S.PRURITUS	Juckreizstudie
S.BIOMATERIAL	Biomaterialstudie

U_S

StudyID	PersID	Type
S.PRURITUS	forster	Other
S.PRURITUS	musterfrau	Investigator
S.BIOMATERIAL	musterfrau	Lab

2.2.3 HTML-Formulare in der Präsentation

Die Präsentationsschicht stellt die Schnittstelle zu den Benutzern eines Anwendungssystems bereit. Im Fall von Webanwendungen, die im Webbrowser des Anwenders ablaufen, wird diese häufig durch Hypertext Markup Language (HTML) realisiert. Diese vom Standardisierungsgremium W3C verabschiedete Auszeichnungssprache versieht die Inhalte einer Webseite mit semantischen Informationen in Form von Tags, so dass sie im Browser strukturiert dargestellt werden können. Wenn im Rahmen dieser Arbeit von HTML ohne weitere Versionsbezeichnung gesprochen wird, ist damit der derzeit aktuelle Standard Extensible Hypertext Markup Language (XHTML) in der Version 1.1 [MIA10] gemeint.

Die Darstellung der durch HTML repräsentierten Struktur kann mittels Cascading Stylesheets (CSS) bestimmt werden, einer ebenfalls durch das W3C standardisierten Formatierungssprache [BÇHL11]. Browserseitige Programmierung der Anwendung erfolgt mittels JavaScript [Ecm11]. Hiermit kann das Verhalten der Präsentationsschicht programmiert werden. In praxi variiert der Einsatz von JavaScript stark. Einfachere Anwendungen setzen es ein, um optionale Unterstützung der Benutzerführung zu realisieren, umfangreichere Anwendungen, wie im Web 2.0 [VH07] üblich, verwenden es, um desktopähnliche Anwendungen als Webanwendung zu realisieren.

Für die in Geschäftsanwendungen häufig benötigte Erfassung und Verarbeitung von Daten stellt HTML Formular-spezifische Funktionen bereit. Mit diesen HTML-Elementen werden Formulare strukturiert und ihr Verhalten definiert. Es ergibt sich eine auf Schlüssel/Wert-Paaren aufbauende Datenstruktur, das Datenmodell der Anwendung wird also durch eine Menge von Paaren aus Attributschlüssel und Attributwert repräsentiert. In Programm 2.1 wird der HTML-Code für ein Formular, das der Bearbeitung von Nutzerdaten in einer Studie dient, gezeigt. Die Darstellung durch einen Webbrowser ist in Abbildung 2.4 gezeigt. Das umfassende Element `form` in Zeile 1 gibt in den Attributen `action` und `method` an, wohin und mit welcher Methode die enthaltenen Formulardaten übertragen werden. Die Formularelemente für die Eingabe der Nutzerdaten werden aus den Elementen `input` und `select` generiert. Ersteres wird als Texteingabefeld dargestellt, während letzteres eine Dropdown-Liste zur Auswahl der angegebenen Optionen erzeugt. HTML sieht die enge Bindung von Daten an die Darstellungselemente vor. So wird ein Schlüssel dadurch erzeugt, dass ein Eingabeelement `input` vorhanden ist und dessen Attribut `name` als Schlüssel verwendet. Beim Absenden des Formulars wird diesem Schlüssel der im Eingabefeld befindliche Wert zugeordnet.

```
<form action="admindata/User" method="post">
  OID<br/>
  <input type="text" name="oid"/><br/>
  Type<br/>
  <select type="text" name="type">
    <option>Sponsor</option>
    <option>Investigator</option>
    <option>Lab</option>
    <option>Other</option>
  </select><br/>
  First Name<br/>
  <input type="text" name="FirstName"/><br/>
  Last Name<br/>
  <input type="text" name="LastName"/><br/>
  Email<br/>
  <input type="text" name="EMail"/><br/>
  <input type="submit" value="Save"/>
  <input type="reset" value="Reset"/>
</form>
```

Programm 2.1: HTML-Code eines Formulars, das die Eingabe von Nutzerdaten ermöglicht. In dieser Form dient es für die Neuanlage eines Nutzers, da es ohne vorbelegte Formularwerte erzeugt wird.

Die Schlüssel/Wert-Paare, die beim Betätigen des Save-Buttons an die Geschäftslogik der Webanwendung gesendet werden, sind in Tabelle 2.2 abgebildet.

Eine weitergehende Strukturierung ist mit reinem HTML nicht möglich, mittels JavaScript lassen sich allerdings komplexere Datenstrukturen programmatisch erzeugen.

Zur Entwicklung der Präsentationsschicht mit HTML, CSS und JavaScript existiert umfangreiche Literatur, beispielsweise [GM11; Hog11; RB12].

2.2.4 Webservices zur Kommunikation

Moderne betriebliche Anwendungen, einschließlich Webanwendungen, sind nur in seltenen Fällen Insellösungen, die isoliert von anderen Systemen arbeiten. Vielmehr handelt es sich um Systeme, die in bestehende Systeme eingebunden werden und auf deren Daten und Funktionen zugreifen bzw. diesen eigene Da-

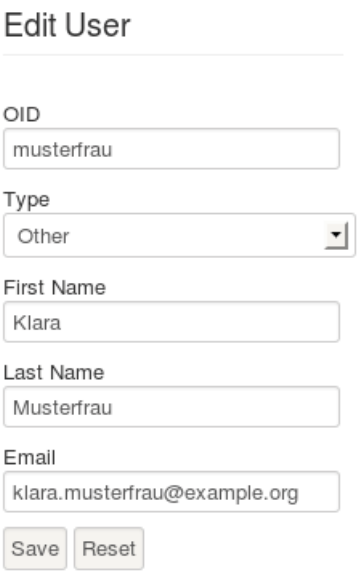


Abbildung 2.4: Darstellung des Eingabeformulars für User-Daten, dessen HTML-Code in Programm 2.1 zu sehen ist. Es wurden bereits Eingaben getätigt.

Tabelle 2.2: Schlüssel/Wert-Paare, die beim Absenden des Formulars aus Abbildung 2.4 an die Geschäftslogik gesendet werden.

Schlüssel	Wert
oid	musterfrau
type	Other
FirstName	Klara
LastName	Musterfrau
EMail	klara.musterfrau@example.org

ten und Funktionen bereitstellen. Die Integration kann sogar so weit gehen, dass die Abgrenzung einzelner Systeme schwierig wird [Has00].

Aus technischer Sicht kann die Integration auf unterschiedlichen Ebenen einer Schichtenarchitektur ansetzen. Für eine Drei-Schichten-Architektur, die Datenhaltung, Geschäftslogik und Präsentation in jeweils eigenen Schichten realisiert, werden die Eigenschaften der unterschiedlichen Integrationsszenarien im Folgenden kurz beschrieben [RMB01].

Bei der Integration auf Datenebene greift ein System direkt auf die Daten eines anderen Systems zu, ohne die Geschäftslogik und Präsentationsschicht zu durchlaufen. Hierbei werden die auf den Daten durchführbaren Aktionen nicht weiter eingeschränkt, allerdings muss die gesamte Funktionalität der Geschäftslogik und der Präsentation eigens implementiert werden.

Bei der funktionalen Integration wird die Geschäftslogik des Systems verwendet, so dass hierbei die dort vorgesehenen Aktionen durchgeführt werden können. Dies schränkt die Möglichkeiten im Vergleich zur Integration auf Datenebene ein, bietet aber Zugriff auf vorhandene Funktionalität.

Bei der Integration auf Präsentationsebene wird auf die Präsentationsschicht des Systems zugegriffen. Dabei können nur die für den Benutzer vorgesehenen Vorgänge ausgeführt werden, die tendenziell von geringer Komplexität sind und durch die zusätzliche Schicht häufig wenig performant.

In der vorliegenden Arbeit wird die Integration auf Ebene der Geschäftslogik am Beispiel von *Webservices* [BHM+04] behandelt. Die Integration auf dieser Ebene bietet viele Vorteile. So wird der Datenzugriff gekapselt, was Entwicklungsaufwand spart und den Zugriff auf die Daten nur mittels erlaubter und passend entwickelter Funktionen der Geschäftslogik zulässt. Webservices stellen eine für diese Integration häufig verwendete Technik dar, die auf Standards des World Wide Web aufsetzt. Für den Begriff Webservice hat sich bisher keine allgemein anerkannte Definition etabliert. Allgemeinen wird darunter ein Dienst verstanden, der von anderen Anwendungen über das Web genutzt werden kann [ACKM04]. Von den historisch älteren, SOAP-basierten Webservices sind die nach dem Prinzip REST entworfenen Webservices zu unterscheiden [Ull12]. Im Folgenden werden beide Arten beschrieben und diskutiert.

SOAP-basierte Webservices SOAP-basierte Webservices gehen auf die Standardisierungsbemühungen des World Wide Web Consortium (W3C) unter dem Titel *Web Services Activity* zurück [36]. Zu diesem Zweck wurde auch eine genauere Definition von Webservices vorgenommen:

> „A Web service is a software system designed to support inter-

> operable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [HB04]

Kern von SOAP-basierten Webservices ist die Zugänglichmachung von Diensten für Anwendungen durch den Austausch von Nachrichten im SOAP-Format [GHM+07], einem XML-basierten Transportformat für die mit dem Webservice auszutauschenden Daten, das typischerweise über das Protokoll *Hypertext Transfer Protocol (HTTP)* übertragen wird.

Der Webservice-Konsument schickt ein SOAP-Dokument als Anfrage und erhält vom Webservice-Anbieter ein SOAP-Dokument mit der Antwort. Abbildung 2.5 zeigt das Prinzip, angelehnt an [Pap08]. Zuerst erzeugt die Anwendung, die als Nutzer des Webservices auftritt, eine Anfrage im SOAP-Format (1). Diese wird über ein Transportmedium übertragen, hier über ein Netzwerk und das HTTP-Protokoll. Der Transportweg zum Webservice-Anbieter kann in eine andere organisatorischen Einheit des Netzwerks führen oder auch via Internet durch fremde Netzwerke. Nach dem Eintreffen der Nachricht beim Anbieter wird diese durch die SOAP-Infrastruktur entgegengenommen und in die applikationsspezifische Sprache konvertiert (2). Sodann wird die Anfrage an die zur Bearbeitung vorgesehene Komponente der Implementierung weitergeleitet (3). Dort wird durch die Geschäftslogik eine entsprechende Antwort generiert und zurückgeliefert (4). Die Antwort wird in das SOAP-Format konvertiert und für den Transport an den SOAP-Server übergeben (5). Nach der Übertragung durch die Transportschicht erreicht die Antwort die Client-Anwendung (6).

Die SOAP-Dokumente eines Nachrichtenaustauschs für das Abfragen von Personendaten sind in Programm 2.2 gezeigt. Die anwendungsspezifischen Nachrichten werden dabei innerhalb eines Elements `Body` transportiert. Dieses ist Bestandteil des durch die SOAP-Spezifikation festgelegten SOAP Envelope-Schemas[2]. Die Zeilen 1 bis 8 enthalten die Anfrage der Client-Anwendung an den Webserver. Zeile 4 enthält ein Token zur Identifizierung des zuvor angemeldeten Clients und Zeile 5 enthält die eigentlich Anfrage. Die Antwort erfolgt durch die Nachricht in den Zeilen 10 bis 24. Die Zeilen 15 bis 19 enthalten das XML-Dokument mit den angefragten Daten.

Die Beschreibung eines Webservices erfolgt in maschinenlesbarer Art gemäß dem Standard Web Services Description Language (WSDL) [CMRW07]. Die

[2] http://www.w3.org/2003/05/soap-envelope/

```
1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.
       org/soap/envelope/" xmlns:urn="urn:exist">
2    <soapenv:Body>
3      <urn:xquery>
4        <urn:sessionId>4833510</urn:sessionId>
5        <urn:xquery>declare namespace odm="http://www.cdisc
             .org/ns/odm/v1.3"; //odm:User[@OID = "
             musterfrau"]</urn:xquery>
6      </urn:xquery>
7    </soapenv:Body>
8  </soapenv:Envelope>
9
10 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.
       org/soap/envelope/">
11   <soapenv:Body>
12     <retrieveDataResponse xmlns="urn:exist">
13       <retrieveDataReturn>
14         <elements>
15           <odm:User xmlns:odm="http://www.cdisc.org/ns/
                 odm/v1.3" OID="musterfrau" UserType="Other"
                 >
16           <odm:FirstName>Klara</odm:FirstName>
17           <odm:LastName>Musterfrau</odm:LastName>
18           <odm:Email>klara.musterfrau@example.org</
                 odm:Email>
19           </odm:User>
20         </elements>
21       </retrieveDataReturn>
22     </retrieveDataResponse>
23   </soapenv:Body>
24 </soapenv:Envelope>
```

Programm 2.2: Ein SOAP-Nachrichtenaustausch zur Übermittlung von Daten der Person mit der Kennung `musterfrau`. Die Anfrage befindet sich in den Zeilen 1 bis 8, die Antwort in den Zeilen 10 bis 24.

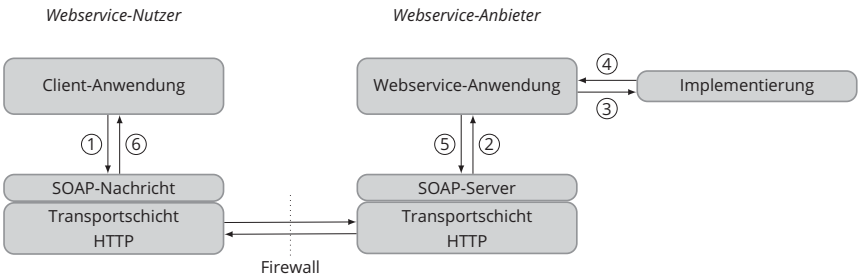


Abbildung 2.5: Kommunikation mit einem SOAP-basierten Webservice, vgl. [Pap08].

ebenfalls XML-basierten WSDL-Dokumente beschreiben, *welche* Dienste angeboten werden und *wie* Nachfrager auf diese zugreifen können. Es können sowohl öffentlich verfügbare als auch unternehmensinterne Dienste eingebunden bzw. angeboten werden. Für das Anbieten und Auffinden von Diensten wurde ein Universal Description, Discovery and Integration (UDDI) genannter Verzeichnisdienst spezifiziert [BDE+02], der in der Praxis allerdings keine wichtige Rolle spielt [25].

SOAP-basierte Webservices und die weiteren genannten Techniken werden auch als WS-* oder „Big" Webservices bezeichnet [PZL08]. Eine weiterführende Darstellung von SOAP-basierten Webservices ist bspw. in [ACKM04] und [WCL+05] gegeben. Eine Erweiterung von XQuery zur Integration von WSDL-basierten Webservices ist in [OS04] beschrieben, aber bisher nicht standardisiert worden.

REST-basierte Webservices Die auf dem Prinzip Representational State Transfer (REST) [Fie00] basierenden Webservices orientieren sich am Verständnis von Ressourcen, die über das HTTP-Protokoll adressiert werden. Das Format der Ressourcen ist nicht vorgegeben, es kann aber XML verwendet werden. Beim Einsatz REST-basierter Webservices gibt es weniger Vorgaben als bei SOAP-basierten. Eine Uniform Resource Locator (URL) verweist auf eine Ressource, die mittels der HTTP-Befehle manipuliert wird. HTTP erfüllt bei REST nicht nur die Rolle des Transportprotokolls, sondern ist als Anwendungsprotokoll obligatorisch. Der Befehl `GET` ruft die durch eine URL referenzierte Ressource ab, `POST` und `PUT` übermitteln die Erzeugung und Veränderung einer Ressource und `DELETE` löscht die Ressource. Die Kommunikation mit dem Server ist dabei zustandslos, d. h. sämtliche für das Ausführen des Befehls notwendige Informatio-

nen werden mit jedem Aufruf übertragen. Die Verwendung von HTTP ermöglicht die Nutzung von Web-Techniken zur Implementierung, wobei aufgrund der Semantik der HTTP-Befehle Routing- und Cachingentscheidungen auf Ebene des Transportprotokolls getroffen werden können. HTTP-Operationen, die, wie `GET`, als „safe" gelten, führen keine Änderungen an der Ressource aus und können somit aus einem Cache beantwortet werden. Bei HTTP-Operationen, die, wie `PUT` oder `DELETE`, als „idempotent" gelten, ändert die mehrfache Ausführung derselben Operation nicht das Ergebnis der erstmaligen Ausführung [Ra07]. Idempotente Befehle können bei vermutetem Verlust der Anfrage, z. B. aufgrund langer Wartezeiten, einfach erneut gesendet werden, ohne das Resultat der Ausführung zu beeinträchtigen.

Das Übertragungsformat der Ressource wird bei REST nicht durch das Protokoll vorgegeben und ist anwendungsspezifisch. Die Verwendung von XML ist möglich. Eine standardisierte Beschreibung des REST-basierten Webservices analog zu WSDL ist nicht vorgesehen. Ebenso gibt es kein standardisiertes Format für einen REST-Verzeichnisdienst.

Diskussion von SOAP und REST In der Literatur werden unterschiedliche Meinungen über die Vorteilhaftigkeit von SOAP und REST vertreten. [MNS05] konstatiert im Jahr 2005, dass neben den technischen Eigenschaften auch entscheidend ist, welcher Standard sich langfristig als dominierend herausstellt. Die generelle Dominanz eines Standards kann allerdings derzeit nicht festgestellt werden. [LW07] argumentiert, dass der REST-Stil aufgrund seiner aus der Vernetzung stammenden Protokolle für vernetzte Anwendungen besser geeignet ist. Lokale Anwendungen seien besser mittels SOAP unter Verwendung von Middleware-Plattformen für den Nachrichtenaustausch anstatt HTTP zu integrieren. [PZL08] resümiert, dass REST als Methode zur Ad-hoc-Integration geeigneter sei, während SOAP mittels WS-* die richtige Technik für langlebige Anwendungen mit hohen Qualitätsanforderungen sei. [BSBG12] zeigt in einer Studie Geschwindigkeitsvorteile von REST, wobei diese sicher nicht ohne Weiteres verallgemeinerbar ist. Zu demselben Ergebnis kommt [MG09] bei der Evaluierung eines Middleware-Frameworks, das SOAP- und REST-Schnittstellen bietet.

Festzuhalten bleibt, dass die Bereitstellung von SOAP-basierten Webservices einen höheren Implementierungsaufwand erfordert, da das Protokoll umfangreicher ist. Damit lassen sich allerdings auch flexiblere Services implementieren. REST-Webservices sind einfacher zu implementieren, da sie auf HTTP aufbauen und mit dessen Befehlen operieren. Geschwindigkeit und Qualitätsmerkmale sind im Allgemeinen implementierungsspezifisch, so dass hieraus keine allge-

meine Empfehlung abgeleitet werden kann.

2.2.5 Softwarestack aus genannten Techniken

Um ein aus den genannten Schichten bestehendes System zu realisieren, müssen sie kompatible Schnittstellen aufweisen. Dies geschieht entweder, indem zueinander nativ kompatible Paradigmen und Techniken eingesetzt werden, oder häufiger, indem die jeweils beste Lösung für eine Schicht gewählt wird und die dadurch entstandene Auswahl an Techniken durch Zwischenschichten kompatibel gemacht wird. Fällt die Wahl für die Geschäftslogik, die über Webservices mit anderen Anwendungen kommunizieren soll, auf Objektorientierung, wird für die Datenschicht ein relationales DBMS verwendet und soll die Präsentation als Webanwendung gestaltet werden, so ergeben sich mehrere Übergänge. Die relationale Datenbank muss an die objektorientierte Geschäftslogik angepasst werden und diese muss Daten für die auf Schlüssel/Wert-Paaren basierende Präsentationsschicht liefern. Für das Beispiel der Benutzerverwaltung, deren objektorientiertes Modell der Geschäftslogik in den entsprechenden Klassen der Abbildung 2.2, das relationale Modell der Persistenzschicht in Abbildung 2.3, das Schlüssel/Wert-Paar-basierte Modell der Präsentationsschicht in Tabelle 2.2 und die per Webservice abrufbare XML-Repräsentation in Programm 2.2 zu finden ist, ergeben sich folgende Transformationen, die erstellt, getestet und gepflegt werden müssen:

- Mappen der Relationenschemata und Klassen
- Mappen der Klassen und Schlüssel/Wert-Paare
- Mappen der Klassen und der XML-Repräsentation

Allgemein existieren so bei n Attributen $3n$ Mappings, die jeweils für beide Transformationsrichtungen benötigt werden. Insgesamt werden also $6n$ Mappings benötigt. Für die Attribute des Datensatzes zu einem Benutzer `oid`, `type`, `FirstName`, `LastName` und `EMail` würden also 30 Mappingaktionen benötigt.

Da die Erzeugung dieser Transformationen von Hand somit sehr aufwändig ist, kommen hierzu meist Frameworks zum Einsatz. Im Fall des Mappings zur Datenschicht sind dies objektrelationale Mapper, beim Übergang von der Geschäftslogik zur Präsentationsschicht sind dies Webanwendungsframeworks. Für Java seien hier beispielhaft die Frameworks Hibernate und Apache Tapestry genannt. Für die Bereitstellung und Einbindung von Webservices stellt Java mit Java API for XML Web Services (JAX-WS) ein Interface bereit, das das Erzeugen

von Webservice-Servern und -Clients erleichtert. Da SOAP-Nachrichten XML-basiert sind, findet auch hierbei ein Mapping des Dokumenteninhalts auf die objektorientierte Geschäftslogik statt.

Hibernate realisiert ein Mapping zwischen den Tabellen eines relationalen Datenbanksystems und den Objekten der darüber liegenden Schicht. Wenn ein Objekt der Geschäftslogik, beispielsweise ein Objekt, das ein Formular repräsentiert, gespeichert werden soll, transformiert Hibernate das Objekt in eine Darstellungsweise, die in der Tabellenstruktur abbildbar ist und speichert den Zustand im relationalen Datenbanksystem. Beim Laden des Formulars rekonstruiert Hibernate aus der Relation wieder das Formularobjekt. Apache Tapestry unterstützt die Bindung der Objekte an Komponenten der HTML-basierten Präsentationsschicht. Damit können beispielsweise Attribute eines Formularobjekts an ein entsprechendes HTML-Formular gebunden werden. Das Framework sorgt dafür, dass die Attribute zur Anzeige des Formulars in Schlüssel/Wert-Paare umgewandelt werden und beim Auslösen der Absenden-Aktion des Formulars durch den Benutzer aus diesen Schlüssel/Wert-Paaren der neue Objektzustand konstruiert wird, um in der Geschäftslogik weiterverarbeitet zu werden. Abbildung 2.6 zeigt den Aufbau einer solchen Drei-Schichten-Architektur als Ausprägung der allgemeinen Schichtenarchitektur [LD04] und verdeutlicht die Position von Mappern, die hier hellgrau mit gestricheltem Rand dargestellt sind.

2.3 Kritik an Paradigmenbrüchen und deren Überbrückungstechniken

Der gezeigte Aufbau führt einerseits dazu, dass die für jede Schicht jeweils am weitesten entwickelten Techniken verwendet werden können, andererseits bewirkt er auch Brüche in der Architektur, die mit Hilfe von Mappern überbrückt werden müssen. Breite Kritik an der Architektur von Webentwicklungen findet sich in [MT07]. In [LSV12] wird Kritik am heterogenen Aufbau der Schichtenarchitektur angedeutet, allerdings fokussiert diese auf das Mischen deklarativer und imperativer Sprachkonzepte. Im Folgenden werden die Problemfelder, mit denen sich die vorliegende Arbeit beschäftigt, herausgestellt.

2.3.1 Objektorientierung und Relationenschema

Allgemein wird Inkompatibilität zwischen der für Datenzugriff verwendeten „sublanguage“ und der für die Programmierung der Geschäftslogik verwendeten „main language“ als „impedance mismatch“ bezeichnet [MS86]. In jüngerer

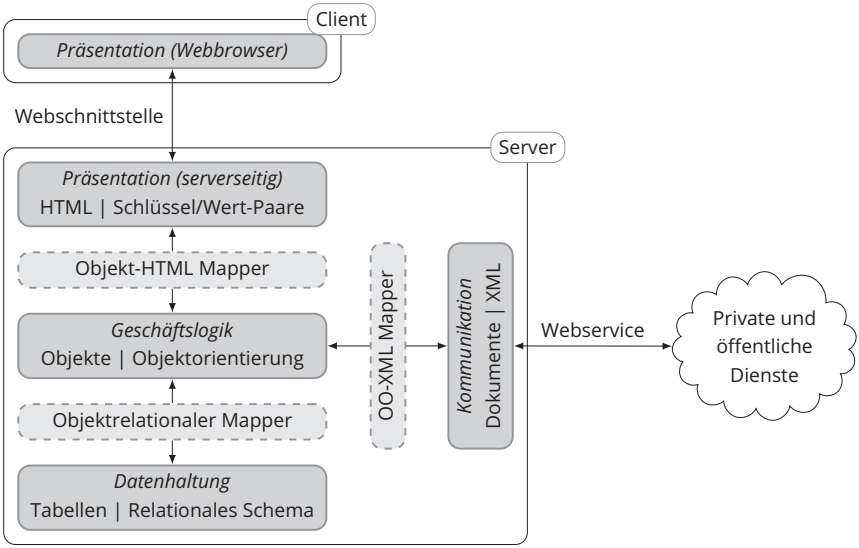


Abbildung 2.6: Gängige Softwarearchitektur, bei der die konzeptionellen Unterschiede zwischen den Schichten mit Mappern überbrückt werden.

Zeit hat sich dieser Begriff für die Bezeichnung des Bruchs zwischen der relationalen Datenhaltungsschicht und der objektorientierten Geschäftslogik etabliert [Ban88; BGD97; CI05; ONe08; IBNW11]. Im Rahmen dieser Arbeit wird *Impedance Mismatch* in der letztgenannten, spezielleren Bedeutung verwendet. Der Softwarearchitekt Ted Neward bezeichnet den Impedance Mismatch in Anlehnung an den Vietnamkrieg als Vietnam der Informatik [27], für das es keine dominante Lösungsstrategie gibt. Seine Problembeschreibung und Lösungsstrategien sind im Folgenden wiedergegeben.

Die naheliegende Überlegung, Klassen des objektorientierten Modells auf Entity-Typen und deren Eigenschaften auf Attribute abzubilden, führt bei Vererbungsbeziehungen zu Problemen. Wie in Abschnitt 2.2.2 angedeutet, existiert im Relationenmodell kein natives Konzept für Vererbung und deren ERM-Entsprechung IS-A. Die Regel, jedes Entity als Tabelle abzubilden und dann IS-A-Beziehungen durch die Inklusion von Schlüsseln zu lösen („table-per-class“), führt zu einem Datenbankschema, das entsprechend viele Abhängigkeiten aufweist und somit beim Laden der Objekte auf teure *Join*-Operationen [ME92] zurückgreifen muss. Um diese zu vermeiden, existieren noch die Strategien, für

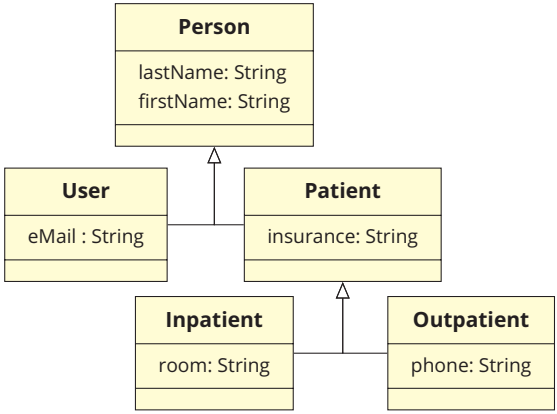


Abbildung 2.7: Vererbungshierarchie der Klasse Person und ihrer Unterklassen.

jeden erzeugbaren Entity-Typen einer IS-A-Beziehung eine vollständige Relation mit allen Attributen („table-per-concrete-class“) zu erstellen sowie die, eine einzige Relation mit allen Attributen, die im Rahmen der IS-A-Beziehung auftreten („table-per-class-family“), anzulegen. Auch diese beiden sind aus Datenbanksicht problematisch, da sie Redundanzen und naturgemäß viele Nullwerte enthalten.

Die Eigenschaften dieser drei Strategien sind in [Fow03] beschrieben und diskutiert. Im Folgenden wird diese Diskussion anhand des in Abbildung 2.7 gezeigten Klassendiagramms, das den aus Abbildung 2.3 bekannten Entity-Typ Person weiter verfeinert, wiedergegeben und es werden die Möglichkeiten und deren jeweilige Problematik bei der Umwandlung von Vererbungsbeziehungen in Relationenschemata illustriert.

In Abbildung 2.8 wird ein zu dem Klassendiagramm passendes ERM-Diagramm gezeigt, das die Grundlage für die Persistierung der Objekte darstellt. Die Vererbungen sind als disjunkte, partielle IS-A-Beziehungen modelliert.

Das nach dem „table-per-class“-Ansatz daraus erzeugte Relationenmodell ist in Tabelle 2.3 gezeigt. Es orientiert sich eng an der objektorientierten Betrachtungsweise, da Eigenschaften, die in einer Spezialisierungsebene hinzukommen, in den Attributen der Relation auf dieser Ebene gespeichert werden. Diese Umsetzung ist redundanzfrei und vermeidet nullwertige Attribute. Allerdings erfordert sie, beim Laden eines Objekts Join-Operationen durchzuführen, um die Eigenschaften eines Objekts aus den Attributen der Datenbank zu rekonstruie-

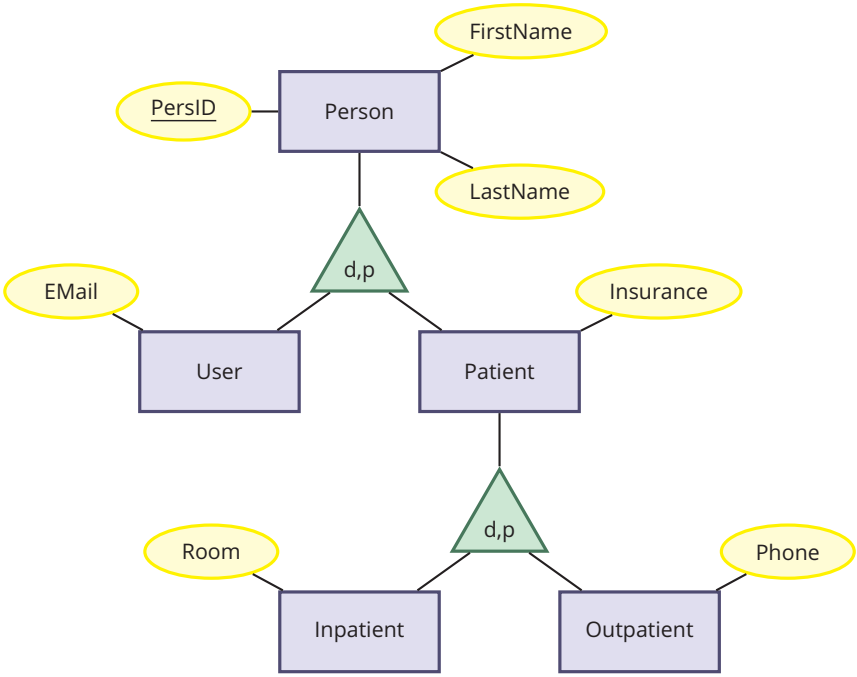


Abbildung 2.8: ERM-Darstellung des Klassendiagramms aus Abbildung 2.7.

ren. So werden beim Laden eines Objektes Join-Operationen über alle Tabellen ausgeführt, die Eigenschaften der übergeordneten Klassen enthalten. Besonders bei Suchanfragen entstehen dadurch hohe Kosten, da sämtliche über- und untergeordnete Objekte der Vererbungshierarchie geladen werden müssen, um sie auf das Suchkriterium zu untersuchen. Die Ansätze „table-per-concrete-class“ und „table-per-class-family“ vermeiden dies:

Die Verringerung der Kosten des Ansatzes „table-per-concrete-class“ wird dabei mit einer Denormalisierung des Datenbankschemas erkauft. Diese Art der Umsetzung des Beispiels ist in Tabelle 2.4 zu finden. Attribute mit gleicher Bedeutung sind darin auf unterschiedliche Relationenschemata verteilt. Dies führt zu einem erhöhten Erstellungs- und Wartungsaufwand, da es dem Prinzip Don’t Repeat Yourself (DRY) [HT99] widerspricht. DRY fordert, dass jeder Sachverhalt nur eine, verbindliche Repräsentation im System haben soll. Durch die redundant vorgehaltenen Attribute erhöht sich die Anzahl der benötigten Mappings.

Tabelle 2.3: Relationenschemata des in Abbildung 2.7 gezeigten ERM-Diagramms, erzeugt gemäß den Transformationsregeln aus Abschnitt 2.2.2 als „table-per-class“.

Person

PersID	LastName	FirstName
forster	Forster	Christian
mustermann	Mustermann	Max
musterfrau	Musterfrau	Klara
doe	Doe	John
roe	Roe	Jane

User

PersID	EMail
forster	christian.forster@ercis.de
mustermann	max@example.org
musterfrau	klara.musterfrau@example.org

Patient

PersID	Insurance
doe	KK1
roe	KK2

Inpatient

PersID	Room
roe	42

Outpatient

PersID	Phone

Außerdem verkompliziert sich die Erzeugung der Primärschlüssel, da sie so erzeugt werden, dass sie Primärschlüssel für alle Tabellen der Vererbungshierarchie sind. Ansonsten könnten beim Laden eines Objekts der Superklasse via Primärschlüssel mehrere Objekte aus unterschiedlichen Unterklassen gefunden werden.

Bei Verwendung des „table-per-class-family“-Ansatzes wird ein einziges Relationenschema verwendet, das sämtliche in der Vererbungshierarche vorkommenden Attribute sowie ein weiteres Attribut, das die Spezialisierung der Oberklasse angibt, enthält. In dem in Tabelle 2.5 gezeigten Relationenschema ist dies das Attribut `PersType`. Der Ansatz ähnelt dem in der Literatur als Universalrelation [Ull88] bezeichneten Entwurf und weist teilweise dessen Probleme auf. Nachteilig sind bei diesem Ansatz insbesondere die planmäßig auftretenden und ggf. indizierten Nullwerte in allen Attributen, die nicht zur jeweiligen Spezialisierung des persistierten Objektes gehören. Diese führen zu Effizienzproblemen

Tabelle 2.4: Relationenschemata des in Abbildung 2.7 gezeigten ERM-Diagramms, das nach dem „table-per-concrete-class“-Ansatz für alle instanziierbaren Klassen Relationen mit allen Attributen enthält.

Person

PersID	LastName	FirstName

User

PersID	LastName	FirstName	EMail
forster	Forster	Christian	christian.forster@ercis.de
mustermann	Mustermann	Max	max@example.org
musterfrau	Musterfrau	Klara	klara.musterfrau@example.org

Patient

PersID	LastName	FirstName	Insurance
doe	Doe	John	KK1

Inpatient

PersID	LastName	FirstName	Insurance	Room
roe	Roe	Jane	KK2	42

Outpatient

PersID	LastName	FirstName	Insurance	Phone

Tabelle 2.5: Relationenschema des in Abbildung 2.7 gezeigten ERM-Diagramms, das gemäß des „table-per-class-family"-Ansatzes eine einzige Relation für alle Klassen enthält. Nullwerte werden durch das Zeichen – dargestellt. Die Tabelle ist zur besseren Übersicht transponiert dargestellt.

PersID	forster	mustermann	musterfrau	doe	roe
PersType	User	User	User	Patient	Inpatient
LastName	Forster	Mustermann	Musterfrau	Doe	Roe
FirstName	Christian	Max	Klara	John	Jane
EMail	christian...@...	max@...	klara....@...	–	–
Insurance	–	–	–	KK1	KK2
Room	–	–	–	–	42
Phone	–	–	–	–	–

bei Relationen mit vielen Tupeln [Vos86].

Jede der gezeigten Abbildungsmöglichkeiten des objektorientierten auf ein relationales Modell weist spezifische Mängel auf, wobei versucht werden kann, durch Kombination der Ansätze deren Auswirkung möglichst gering zu halten. Objektrelationale DBMS, bei denen das relationale Modell um objektorientierte Funktionen erweitert wird, verbergen diese Abbildungen vor dem Benutzer, indem sie objektorientierte Erweiterungen von SQL verwenden. Über entsprechende SQL-Befehle können benutzerdefinierte Datentyp-Hierarchien definiert, typisierte Tabellen dafür angelegt und Objekte erzeugt, gelesen, geändert und gelöscht werden. Intern bilden objektrelationale DBMS die Objektstruktur auf ein relationales Modell ab, wodurch wiederum die Nachteile des jeweiligen Ansatzes in Kauf genommen werden müssen. Dies geschieht bspw. bei den objektrelationalen DBMS *DB2* von *IBM*[3] [CCN+99] und dem *Database Server* von *Oracle*[4] [BG08]. *Microsoft* stellt bei seinem DBMS *SQL Server*[5] keine objektrelationale Schnittstelle bereit, sondern verlagert das objektrelationale Mapping in die übergeordnete Schicht. Da mit der Sprache *Language Integrated Query (LINQ)* eine integrierte Abfragesprache zur Verfügung steht, wird zwar die Inkompatibilität zwischen main language und sublanguage gemindert, was der Forderung nach einer erweiterten Datenbank Application Programming Interface (API) entgegenkommt [ZR01]. Der konzeptionelle Unterschied besteht allerdings weiterhin und muss durch Anwendung der bereits erwähnten Map-

[3] http://www-01.ibm.com/software/data/db2/

[4] http://www.oracle.com/us/products/database/overview/index.html

[5] https://www.microsoft.com/en-us/sqlserver/default.aspx

pingstrategien überbrückt werden [13].

Folgende Strategien bieten sich zum Umgang mit diesem objektrelationalen Paradigmenbruch an [27]:

1. Aufgabe der Objektorientierung der Geschäftslogik und Beibehalten des relationalen Schemas in der Persistenzschicht
2. Aufgabe des relationalen Schemas der Persistenzschicht und Verwendung einer objektorientierten Datenbank
3. Manuelles Mappen jeder Klasse in entsprechende Relationen
4. Akzeptieren der Limitationen von objektrelationalen Mappern
5. Integration von relationalen Konzepten in die Programmiersprache
6. Integration von relationalen Konzepten in das Framework

Jede dieser Strategien ist mit Einschränkungen verbunden. Ansatz 1 bedeutet mit dem Verzicht auf Objektorientierung eine Abkehr vom derzeitigen Standardvorgehen der Softwareentwicklung. Folgt man Ansatz 2 und gibt die relationale Persistenzschicht zu Gunsten einer objektorientierten auf, so muss in Kauf genommen werden, dass die verfügbaren objektorientierten Datenbankmanagementsysteme sich nicht als „generell tragfähig" [Vos08] erwiesen haben, hinter den Erwartungen zurückgeblieben sind [CD96] und weniger verbreitet sind als ihre relationalen Pendants oder Produkte für spezielle Nischenmärkte sind [SV08]. Die Strategie 3 setzt darauf, dass die Entwickler die Zustandsinformationen der Objekte manuell in passende Tabellenstrukturen übertragen und aus diesen lesen, während Strategie 4 dieses Mapping zu automatisieren sucht. In beiden Fällen bleibt der konzeptionelle Bruch bestehen und kann zu ineffizienten Ergebnissen führen. Benchmarks deuten darauf hin, dass der Einsatz von objektrelationalen Mappern im Allgemeinen zu deutlichen Geschwindigkeitsverlusten gegenüber der Verwendung von objektorientierten DBMS führt [ZKB06; Kop08], obwohl diese einen deutlich geringeren Verbreitungs- und Reifegrad als die den Mappern zugrunde liegenden relationalen DBMS aufweisen. Bei Benutzung eines Mappers hat zudem noch die nicht-triviale Konfiguration weiterer Parameter Einfluss auf die Performanz [ZKCB09]. Ferner kann die Verwendung bestimmter objektorientierter Konstrukte wie 1:n-Assoziationen zu Geschwindigkeitseinbußen bei der Persistierung führen [Weg13]. Trotzdem ist die Verwendung objektrelationaler Mapper, wie in Abschnitt 2.2.5 am Beispiel Hibernate beschrieben, in der Praxis häufig. Ansatz 5 baut darauf, objektorientierte Sprachen um relationale Konzepte zu erweitern, allerdings sind die

Erfolge dieser Bemühungen derzeit noch begrenzt [27]. Die letzte Strategie 6 beschreibt die Idee, beim Entwurf der Klassen bereits auf gute Abbildbarkeit auf relationale Konzepte zu achten, indem entsprechende Sprachkonzepte der objektorientierten Sprache verwendet werden.

Der in dieser Arbeit gewählte Ansatz basiert darauf, den Paradigmenkonflikt bereits im Ansatz zu umgehen, indem ein für alle Schichten geeignetes Konzept gewählt wird. Dies entspricht der Denkweise, die Ausweg 1 und 2 zu Grunde liegt, setzt aber anstelle relationaler Schemata und Objektorientierung auf die dokumentenorientierte Sicht von XML und die dazugehörigen Techniken, denen aufgrund der Bedeutung für diese Arbeit ein eigenes Kapitel 3 gewidmet ist.

2.3.2 Objektorientierung und Schlüssel/Wert-Paare

In einer geschäftlichen Webanwendung werden Daten dem Nutzer in Form von Formularen zur Ansicht und Bearbeitung präsentiert. Die dazu in HTML vorgesehenen Formularelemente weisen mit ihrem auf Schlüsse/Wert-Paaren basierenden Datenmodell erneut einen konzeptionellen Unterschied zur Objektorientierung der Geschäftslogik auf.

Allerdings ist dieser Unterschied im Allgemeinen trotz der unterschiedlichen Datenstruktur nicht problematisch, wie im Folgenden argumentiert wird. Die Probleme des objektrelationalen Mappings liegen in der Komplexität der Objektstrukturen und dem Umfang der Daten begründet. Komplexität und Umfang der in der Präsentationsschicht abzubildenden Aspekte der Geschäftsobjekte müssen für die Präsentation im Browser derart beschränkt werden, dass sie angemessen wiedergegeben und vom Anwender leicht erfasst werden können. Somit erzeugen auch die auf Schlüssel/Wert-Paare zu mappenden Eigenschaften der Objekte keine ernsthaften Effizienzprobleme.

Aus praktischer Sicht bedeutet dies, dass das Mapping von Geschäftsobjekten auf Formulare keinen prinzipbedingten Effizienznachteil aufweist. Dieses Mapping und die dazugehörigen HTML-Elemente können sowohl manuell erfolgen als auch durch ein Framework erzeugt werden. Neben Mapping der Daten und Erzeugung der HTML-Formularelemente ist noch das Verhalten des Formulars relevant. Für eine gute Benutzerführung sind hierbei zum Beispiel Hinweise auf fehlende oder fehlerhafte Eingaben, Hilfen zur Dateneingabe, wie z. B. die Anzeige eines Kalenders zur Unterstützung einer Datumseingabe oder das Anzeigen von Erläuterungstexten, sowie das Ein- und Ausblenden von Formularelementen in Abhängigkeit von vorherigen Eingaben nötig. Reine HTML-Formulare beherrschen diese Funktionen nicht und müssen dazu mit JavaScript

ergänzt werden. Solche Verhaltensaspekte können ebenfalls manuell programmiert oder durch das Webanwendungsframework generiert werden.

In dem meisten Fällen wird man auf ein derartiges Framework zurückgreifen, so dass hierfür wenig Entwicklungsaufwand betrieben werden muss. Je nach in der Geschäftslogik eingesetzter objektorientierter Programmiersprache stehen unterschiedliche Webanwendungsframeworks zur Verfügung. Im Fall von Java steht mit JavaServer Faces (JSF) im Rahmen der Enterprise Edition-Plattform eine Spezifikation zur Verfügung, für die es mehrere implementierende Frameworks gibt. Trotzdem findet sich für Java eine Vielzahl weiterer Webanwendungsframeworks in Verwendung, die diesen Standard nicht implementieren und daher spezifische, nichtportable Entwicklungen erfordern.

Der Einsatz von Webanwendungsframeworks hat den Vorteil, dass damit von der Entwicklung in JavaScript und HTML größtenteils abgesehen werden kann. Diese ist nur nötig, wenn Funktionen erstellt werden sollen, die über den Umfang des eingesetzten Frameworks hinausgehen oder eben kein Framework zum Einsatz kommt. Im günstigsten Fall geschieht das Mapping – falls vom Webanwendungsframework unterstützt – automatisiert. Anderenfalls muss sich der Entwickler bei dem Übergang von der serverseitigen Programmiersprache zu JavaScript mit der Transformation von Objekten befassen und bei manuellem Mapping der Objekte auf Schlüssel/Wert-Paare auch mit der Transformation zwischen den beiden Datenmodellen.

2.3.3 Objektorientierung und Dokumentenaustausch

Bei der Kommunikation von Anwendungen mittels Webservices werden Informationen auf Ebene der Geschäftslogik ausgetauscht. Dies geschieht in Form von SOAP-Dokumenten, wobei die inhaltliche Bedeutung dieser nicht durch das Format festgelegt ist. Bei Entwicklung des SOAP-Formats stand die Bezeichnung als Akronym für *Simple Object Access Protocol* und verdeutlichte den Designzweck, nämlich den Zugriff auf Objekte zu ermöglichen. Mittlerweile gilt SOAP als Eigenname, was zutreffend andeutet, dass die Objektorientierung nicht zwangsläufig Bestandteil des SOAP-Konzepts ist. Vielmehr wird mittels SOAP lediglich festgelegt, wie Aufrufe und Antworten in Form von XML-Nachrichten erfolgen. Neben dem RPC-Style genannten, entfernten Methodenaufruf mittels SOAP, bei dem Methodenname und Parameter übergeben werden, können SOAP-Nachrichten im Document-Style nämlich beliebige XML-Dokumente übertragen. Für das Mapping von Daten aus der Programmiersprache, die den Webservice implementiert, auf XML-Elemente sind zwei Vorgehensweisen möglich: Entweder werden Regeln für Serialisierung und Deserialisierung explizit

durch den *encoding style* innerhalb der SOAP-Nachricht angegeben oder sie werden in der WSDL-Datei festgelegt [Pap08].

Die Einbindung von Webservices in die objektorientierte Geschäftslogik stellt somit keinen paradigmatischen Bruch dar. Transformationen durch Mapping sind aber erforderlich, um die XML-basierten Nachrichten mit der objektorientierten Geschäftslogik zu erzeugen und zu verarbeiten.

2.3.4 Zusammenfassung

Die Verwendung der in der jeweiligen Schicht als Beste angesehenen Technik führt zu einer Mischung von Techniken, deren Unterschiede mit Hilfe von Mappern überbrückt werden. Im Fall des objektrelationalen Mappings muss der konzeptionelle Unterschied zwischen den beiden Konzepten unter Einbuße von Effizienz gelöst werden. Beim Übergang von der Geschäftslogik zur Darstellung mit Web-Techniken und bei der Einbindung von Webservices ist keine konzeptioneller, sondern lediglich ein technischer Bruch vorhanden. Wünschenswert ist hingegen ein einheitlicher Ansatz bei der Entwicklung von Webanwendungen, der keine Brüche zwischen den Schichten bewirkt [LSLV11].

2.4 Einordnung in der Literatur

In der Literatur werden Problemdarstellungen und Lösungsansätze für Problematik der Kompatibilität der Schichten je nach betroffenem Übergang behandelt. Diese Inkompatibilitäten werden laut den „Best Practices“ mit objektrelationalen Mappern und Web-Frameworks überbrückt [AHKP09].

Der Bruch zwischen relationaler Persistenzschicht und objektorientierter Geschäftslogik wurde bereits in den 1980er Jahren erkannt und als „impedance mismatch“ charakterisiert [MS86]. Seither finden sich in der Literatur Lösungsvorschläge, die sich an den jeweils populären objektorientierten Programmiersprachen orientieren, so für Smalltalk [CM84; SZ87], C++ [KJA93; Hoh96] und Java [BS98; ONe08]. Losgelöst von konkreten Implementierungen wird das Thema konzeptionell immer wieder diskutiert. Die Diskussion zielt meist auf eine Verbesserung des Mappings ab [Rus08]. Es wird eine detaillierte Modellierung vorgeschlagen, um beim Einsatz vom Mappern flexibel die Mappingstrategie vorzugeben [CC05]. Ein Framework zur Kategorisierung des Impedance Mismatch soll das Finden geeigneter Lösungsstrategien erleichtern [IBNW09]. Andere Ansätze vermeiden den Paradigmenbruch, indem ein objektorientiertes DBMS zur Persistierung der Objekte genutzt wird [Trz13].

Die Thematik der Anbindung von HTML-Formularen in der Darstellungsschicht und deren Unterstützung durch Frameworks ist mit dem Aufkommen von Webanwendungen seit Mitte der 1990er Jahre relevant geworden [Fra99]. Diese decken neben der Unterstützung der Darstellungsschicht meist noch weitere Bereiche wie z. B. Persistenzfunktionen, die auf die zuvor genannten Mappingverfahren aufbauen, ab [SH06]. Die Generierung von HTML-Elementen, insbesondere Formularen, und die Bindung der Objekte an diese mit Mitteln der objektorientierten Sprache werden in Java JSF und dessen Vorgänger JavaServer Pages (JSP) durch Tag Bibliotheken [Sil01] realisiert. Im Ruby-basierten Framework Ruby on Rails existieren für diese Funktionen die *Helper*-Methoden für Formulare [LDG12].

Für die Einbindung von Webservices lässt sich eine breite Softwareunterstützung feststellen. Die Serialisierung und Deserialisierung von Objekten zu XML ist beschrieben [GSW+00] und evaluiert [HJR+03]. Für C++ wird ein Framework beschrieben, mit dem sich einerseits Header-Dateien, also C++ Schnittstellenbeschreibungen, aus WSDL-Dateien generieren lassen und andererseits WSDL-Dateien zu bereits vorhandenen C++ Funktionen erzeugt werden können [Eng08]. Das Mapping von XML-Datentypen auf C++, insbesondere für den Anwendungsfall Webservices, ist ebenfalls beschrieben und implementiert [PDZ09]. Für Java ist die Anbindung von Webservices in JAX-WS [Jit11] spezifiziert, die Serialisierung und Deserialisierung von Objekten in XML-Daten werden in Java Architecture for XML Binding (JAXB) [Koh09] spezifiziert. Beide Spezifikationen sind Bestandteil der Mantelspezifikation Java Enterprise Edition (Java EE) in den derzeit aktuellen Versionen 5 bis 7 [28]. Zudem existieren noch leichtgewichtigere Implementierungen der Spezifikationen, wie Apache CXF[6] oder Apache Axis2/Java[7]. Für Ruby on Rails steht ActiveResource[8] zur Verfügung, womit auf Ressourcen, die aus Webservices stammen, objektorientiert zugegriffen werden kann [MO08].

Es lässt sich feststellen, dass die Problematik der Paradigmenbrüche nicht abschließend gelöst ist. Insbesondere der Übergang von einer relationalen Persistenzschicht zu einer objektorientierten Geschäftslogik ist fundamental schwierig und die implementierungsspezifischen Lösungsvorschläge greifen sämtlich auf eine der theoretisch unbefriedigenden, mit spezifischen Nachteilen behafteten Mappingstrategien zurück. Die Abbildung der Objekte auf die in HTML verfügbaren Techniken zur Realisierung der Darstellungsschicht wird durch

[6] http://cxf.apache.org/

[7] http://axis.apache.org/axis2/java/core/

[8] http://api.rubyonrails.org/files/activeresource/README_rdoc.html

Webframeworks eleganter gelöst. Für die Einbindung von Webservices steht ebenfalls eine vielfältige Softwareunterstützung bereit. Da in den beiden letztgenannten Fällen kein fundamentaler Gegensatz überbrückt werden muss, ist dies auch aus theoretischer Sicht einfacher zu lösen. In allen Fällen bleibt allerdings die Notwendigkeit bestehen, zusätzliche Softwarekomponenten einzusetzen.

2.5 Referenzen auf aktuelle Trends und Paradigmen

In diesem Abschnitt werden aktuelle Entwicklungen, die im Zusammenhang mit der in dieser Arbeit vorgeschlagenen XML-basierten Architektur Synergien bewirken können, vorgestellt.

2.5.1 Agile Entwicklung

Der hier gegebene Überblick zu agiler Softwareentwicklung basiert in weiten Teilen auf [BW08].

Softwareentwicklung verfolgt das Ziel, ein für den Bedarf des Anwenders geeignetes Softwaresystem zu erstellen. Die klassischerweise dazu verwendeten Methoden, bei denen ein anfangs aufgestellter Plan befolgt wird, indem bspw. ein Pflichtenheft abgearbeitet wird, setzen voraus, dass der Bedarf des Anwenders von vornherein bekannt ist und nach dem Transformationsprozess aus Analyse, Entwurf, Implementierung und Inbetriebnahme durch eine nutzbare Software abgedeckt wird. Da der tatsächliche Bedarf des Anwenders vom zuvor festgelegten Bedarf abweichen kann und die Transformationsschritte keine eindeutigen Resultate hervorbringen, kann erst nach Fertigstellung beurteilt werden, ob das erstellte Softwaresystem den Bedarf des Anwenders abdeckt. Bei einem sequentiellen Vorgehen bewirkt ein Änderungsbedarf weitreichende und somit teure Modifikationen.

Agile Methoden setzen darauf, den Kunden in die Entwicklung einzubinden und früh mit bereits funktionierenden Teilen des Softwaresystems zu konfrontieren. Auf diese Weise kann Änderungsbedarf vom Kunden frühzeitig erkannt und rechtzeitig im Entwicklungsprozess berücksichtigt werden.

Die grundlegenden Werte dazu sind im agilen Manifest [11] festgehalten. Es erklärt die folgenden Präferenzen:

- „Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation

- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans“

Die Qualifikation und Erfahrung der Menschen, die mit der Softwareentwicklung befasst sind, und ihre Interaktion zählen im Zweifel also mehr als festgelegte Vorgehensweisen und konkrete Tools, die als Hilfsmittel verstanden werden, um den Entwicklern die Arbeit zu erleichtern, aber sie nicht einschränken sollen. Lauffähige Software wird als wichtiger eingeschätzt als die Beschreibung des Softwareprojekts, da nur anhand der lauffähigen Software der konkrete Nutzen für den Anwender überprüft werden kann. Die Zusammenarbeit mit dem Kunden im Projektverlauf wird als wichtiger bewertet als die a priori stattfindende präzise Beschreibung des Ergebnisses. Da die initiale Aufstellung präziser Anforderungen sich erfahrungsgemäß nicht mit den tatsächlich benötigten Funktionen deckt, ist es wichtiger im Projektverlauf die Kundenwünsche zu berücksichtigen, als ein zwar präzise spezifiziertes und vertraglich vereinbartes, aber tatsächlich weniger nützliches Softwaresystem zu erstellen. Da Änderungsbedarf erst an der laufenden Software erkannt werden kann, ist es wichtiger, auf diese Veränderungen zu reagieren und ggf. den Entwicklungsprozess zu ändern als einen initial festgelegten Projektplan zu verfolgen.

Die oben genannten Werte führen dazu, dass in allen agilen Methoden Software früh und häufig an den Anwender ausgeliefert wird und das Urteil des Anwenders als Rückkopplung auf den Entwicklungsprozess wirkt.

Ein Vorgehensmodell, das diesem Ansatz folgt, ist *Scrum* [Sch95; SB08]. Dabei werden alle Funktionen der zu entwickelnden Software in einem *Product Backlog* gesammelt, um schrittweise im Rahmen von *Sprint* genannten Zeiträumen implementiert zu werden. Für einen Sprint wird eine Menge von Anforderungen aus dem Product Backlog beim *Sprint Planning Meeting* in das *Sprint Backlog* übertragen. Die Priorisierung, welche Funktionen das sind, trifft der *Product Owner*, der die Rolle des Kunden repräsentiert. Das *Team* der Entwickler schätzt den Aufwand der Funktionen und einigt sich mit ihm über die im kommenden Sprint zu bewältigende Menge. Der *Scrum Master* moderiert den Prozess. Er leitet auch die täglich stattfindenden *Daily-Scrum-Meetings*, bei denen die Entwickler einander kurz berichten, was sie seit dem letzten Daily-Scrum-Meeting getan haben, was sie bis zum nächsten Meeting tun werden und welche Hindernisse es gibt. Ein Sprint endet mit dem *Sprint Review*, bei dem das Team dem Product Owner das Ergebnis des Sprints präsentiert. Falls eine Anforderung nicht abgearbeitet werden konnte, wird sie in das Product Backlog zurücksortiert. Das Sprint Review dient auch der Reflexion über den Entwicklungsprozess und ggf.

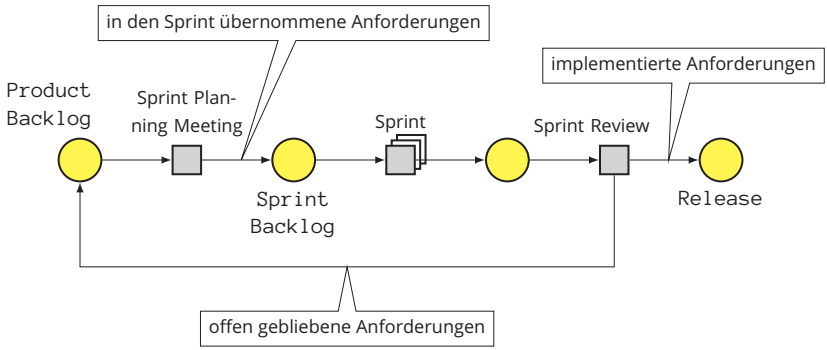


Abbildung 2.9: Der Scrum-Prozess. Die Aktivität Sprint enthält die Durchführung der Implementierungstätigkeit inklusive der Daily-Scrum-Meetings.

der Anpassung des zu bewältigenden Umfangs für den nächsten Sprint. Abbildung 2.9 visualisiert den Scrum-Prozess als XML-Netz, vgl. Abschnitt 3.4.1.

Der Erfolg agiler Methoden hängt von einer Reihe projektspezifischer Parameter ab. So ist es unabdingbar, dass der Kunde vom agilen Vorgehen überzeugt ist und seiner aktiven Rolle gerecht wird. Lange Entscheidungsverfahren seitens des Kunden behindern den Projektfortschritt. Bei lebenskritischen Softwaresystemen, deren Korrektheit nur im Ganzen nachweisbar ist, bringt dem Anwender eine inkrementelle Bereitstellung keinen Nutzen und eine Rückkoppelung ist nicht möglich. Generell wird bei solchen Systemen nicht auf eine umfangreiche Spezifikation verzichtet werden können, so dass ein agiles Vorgehen hierbei ausscheidet. Gut geeignet ist das agile Vorgehen hingegen, wenn ein innovatives Softwaresystem erstellt werden soll, über das noch nicht viel Erfahrung vorliegt, so dass die Rückkopplungen einen wichtigen Wissenszuwachs bewirken. Falls schnell Teilfunktionalität produktiv nutzbar sein soll, ist agiles Vorgehen ebenfalls vorteilhaft, da der Einfluss des Kunden und die kurzen Entwicklungszyklen es ermöglichen, dass diejenigen Funktionen mit dem für den Anwender größten Nutzen zuerst implementiert werden und zur Verfügung stehen.

2.5.2 Serviceorientierung

Die in Abschnitt 2.2.4 vorgestellten Webservices lassen sich nicht nur als technisches Mittel zur Systemintegration verwenden. Unter dem Begriff *Serviceorientierte Architektur (SOA)* wird eine IT-Strategie postuliert, bei der die Verwen-

dung von Diensten ein wesentliches Merkmal ist [LH08]. Eine SOA setzt komplexe Systeme, die Geschäftsprozesse abbilden, aus einfacheren Systemen, die einzelne Dienste oder Teilprozesse abbilden, zusammen. Durch die Dienstorientierung werden Wiederverwendbarkeit einzelner Komponenten, geringe Kopplung der Komponenten untereinander und eine präzise Aufteilung der Verantwortlichkeiten gefördert [NL05]. Wenn die Infrastruktur einer SOA vorhanden ist, kann eine Anwendung für neue Geschäftsprozesse durch *Orchestrierung* der bestehenden Komponenten erstellt werden. Häufig wird die Schnittstelle zwischen den Diensten durch SOAP-basierte Webservices implementiert, die XML verwenden. [Erl05] bezeichnet die Verwendung von XML als entscheidend für eine SOA. XML stellt jedenfalls eine wichtige Schlüsseltechnik für den Datenaustausch einer SOA dar: Das Transportformat *SOAP* ist XML-basiert und XML wird häufig als Dokumentenformat für die per SOAP ausgetauschten Inhalte verwendet. Dies ergibt einen natürlichen Anknüpfungspunkt für die Verwendung eines Softwaresystems mit XML-basierter Architektur als Komponente einer SOA.

3 XML-Techniken für Webanwendungen

Die erweiterbare Auszeichnungssprache XML stellt die Grundlage einer Sammlung von Techniken dar, die durch das Standardisierungsgremium W3C verabschiedet werden und für das Verständnis der folgenden Kapitel hier erläutert werden. XML bietet als Sprachkonzept die Möglichkeit, Daten und deren Strukturierungsmerkmale in maschinenlesbaren Dokumenten zu speichern und auszutauschen. Die Datenebene der vorgestellten Architektur basiert auf XML-Dokumenten. Pfade in XML-Dokumenten lassen sich durch XPath ausdrücken, einem Bestandteil von XQuery. Häufig auf die Funktion als Abfragesprache für XML reduziert, stellt XQuery eine Turing-vollständige, funktionale Programmiersprache dar, von der im Rahmen der vorliegenden Arbeit intensiv Gebrauch gemacht wird. Für die Beschreibung von Formularen in der Darstellungsebene wird auf XForms zurückgegriffen. Die XForms-Architektur separiert Daten, Darstellung und Kontrollfluss. Daten können dabei durch beliebige XML-Dokumente repräsentiert werden, Darstellung und Kontrollfluss werden gemäß dem XForms-Standard in XML beschrieben und durch einen Interpreter in ein Ausgabeformat umgewandelt. Nach diesen für den Kern der XML-basierten Architektur elementaren Begriffen werden weitere relevante Techniken eingeführt: XSLT ist eine Sprache zur Transformation von Quell-XML-Dokumenten in andere, gegebenenfalls ebenfalls XML-basierte, Zieldokumente. Dann wird mit XSL FO eine Seitenbeschreibungssprache zum Schriftsatz vorgestellt, mit der XML-Dokumente insbesondere zur Darstellung als Druckwerk aufbereitet werden können. Schließlich wird mit CDISC ODM ein Standard der klinischen Domäne eingeführt, der auf XML basiert und im Rahmen des Single-Source-Projekts Verwendung findet.

3.1 XML

Mit dem Begriff XML (Extensible Markup Language) werden häufig nicht trennscharf sowohl die Auszeichnungssprache als auch die zugehörigen Techniken bezeichnet. Dieser Abschnitt stellt XML im engeren Sinne, also die Auszeichnungssprache, dar.

3.1.1 Zweck

Die erste und derzeit für den Einsatz relevante[1] XML-Spezifikation 1.0 wurde im Jahr 1996 durch das Standardisierungsgremium W3C veröffentlicht und hat seitdem lediglich redaktionelle Aktualisierungen erfahren. Die bis dato letzte Überarbeitung erfolgte im Jahr 2008 [BPS+08]. Zweck der Entwicklung von XML ist eine textbasierte Auszeichnungssprache, mit der Dokumente strukturiert werden. Einige der in der Spezifikation genannten Entwicklungsziele sind im Folgenden aufgeführt, weil sie die im Rahmen dieser Arbeit erforderliche Universalität von XML widerspiegeln:

- XML soll problemlos im Internet zu verwenden sein.
- XML soll eine große Breite an Einsatzmöglichkeiten unterstützen.
- Es soll einfach sein, Programme zu erstellen, die XML-Dokumente verarbeiten.
- XML-Dokumente sollen einfach erstellbar sein.

Die einfache Verwendbarkeit im Internet bzw. in Netzwerken im Allgemeinen ist für die Entwicklung verteilter Anwendungen und die Verwendung zur Kommunikation wie bei Webservices hilfreich. Als Grundlage für Webanwendungen unterschiedlicher Domänen ist vorteilhaft, dass XML im Hinblick auf breite Einsatzmöglichkeiten entworfen wurde. Die Fokussierung auf Einfachheit bei der Erstellung sowohl von Programmen, die XML verarbeiten, als auch von XML-Dokumenten selbst, unterstützt auch die Komplexitätsreduktion bei der Verwendung von XML als konstituierendes Architekturelement.

Die breite Anwendbarkeit von XML spiegelt sich auch in der Tatsache wider, dass XML-Dokumente sowohl zur Übermittlung von Nachrichten verwendet werden als auch in XML-Datenbanken persistiert werden können. Für die Übermittelung von XML-Dokumenten können beliebige Übertragungsarten gewählt werden, die textuelle Nachrichten übertragen können. Zur Persistierung eignen sich DBMS. Dabei variiert bei diesen der Grad an XML-Unterstützung. Nicht ganz trennscharf wird zwischen nativen XML-DBMS und XML-fähigen DBMS unterschieden. Native XML-DBMS sind speziell mit der Fokussierung auf XML-Dokumente und assoziierte Standards entworfene DBMS. XML-fähige DBMS

[1] Die XML-Spezifikation 1.1 wurde im Jahr 2004 veröffentlicht und erweitert im Wesentlichen die für die Auszeichnung zulässigen Zeichen. Da dies für die meisten Anwendungen irrelevant ist und die Dokumente nicht abwärtskompatibel sind, wird diese Spezifikation in praxi kaum verwendet.

basieren auf einem anderen, meist relationalen Kern und weisen in unterschiedlichem Umfang XML-Fähigkeiten auf. Im Rahmen dieser Arbeit werden XML-DBMS als solche verstanden, die über ein logisches Modell für XML-Dokumente verfügen und dieses über XML-basierte Techniken zugreifbar machen. Dies trifft auf alle nativen XML-DBMS zu und ggf. auf manche der XML-fähigen DBMS. Von der DBMS-internen Repräsentation kann in Bezug auf das hier vorgestellte Architekturkonzept abstrahiert werden, da es aus Sicht der Softwareentwicklung auf die vorhandenen Schnittstellen Bezug nimmt.

3.1.2 Sprachelemente

Das Paradigma von XML ist die Dokumentenorientierung, das heißt, dass in XML repräsentierte Daten als Textdokumente vorliegen, die bestimmten Eigenschaften genügen. Derartige Dokumente heißen *wohlgeformt* und dieses Kriterium ist die Voraussetzung für die Verarbeitung des Dokuments mit XML-Techniken. Ein wohlgeformtes XML-Dokument besteht aus Elementen, die die logische Struktur eines Baumes aufweisen, und somit hierarchisch angeordnet sind. Ein nur aus dem Wurzelelement bestehendes XML-Dokument ist das Einfachste wohlgeformte. Elemente werden in XML benannt und durch Tags genannte Bezeichner dargestellt. Ein Start-Tag beginnt ein XML-Element und ein End-Tag schließt es. Tags werden durch spitze Klammern eingeschlossen und das End-Tag wird durch einen Schrägstrich gekennzeichnet, sodass sich folgendes Erscheinungsbild ergibt:

```
<element></element>
```

Innerhalb der Tags steht der Inhalt des Elements. Er wird durch den Elementnamen semantisch ausgezeichnet. Der Inhalt kann aus weiteren Elementen und Text bestehen. Falls ein Element sowohl weitere Elemente als auch Text enthält, wird der Inhalt als *mixed content* bezeichnet. Durch diese Verschachtelung von Elementen entsteht die hierarchische Struktur des Dokuments, wobei nicht nur die Beziehungen von Untergeordneten und übergeordneten Elementen, sondern auch die Reihenfolge von Knoten innerhalb eines Elements relevant sind. Zusätzlich können einem Element textliche Attribute zugeordnet werden, indem diese innerhalb des Start-Tags als Schlüssel/Wert-Paare stehen. Bei leeren Elementen können das Start- und End-Tag zusammengefasst werden. Ein leeres Element mit einem Attribut hat somit folgende Form:

```
<element attributename="Attributwert" />
```

XML-Dokumente enthalten Daten stets zusammen mit den sie beschreibenden Daten, also den Metadaten, so dass zur Interpretation keine externen Schemainformationen benötigt werden und die Interpretationsfähigkeit auch nicht

von der Konformität mit einem starren Schema abhängt. Daher werden XML-Dokumente auch als *semistrukturierte* Daten bezeichnet [ABS99].

Da beim Austausch von Dokumenten über unterschiedliche Anwendungen hinweg Elementnamen mehrfach mit unterschiedlicher Bedeutung verwendet werden könnten, lassen sich XML-Elemente Namensräumen zuordnen. Die Verwendung von Namensräumen ist optional, aber für Interoperabilität meist sinnvoll. Ein Namensraum ist durch eine eindeutige Zeichenkette in Form eines Uniform Resource Identifier (URI) gekennzeichnet und wird durch das Attribut `xlmns` als voreingestellter Namensraum einem Element und seinen Kindelementen zuordnen, bspw. so:

```
<element xmlns="http:\x4t.uni-muenster.de"/>
```

Die Bedeutung der URI für den Namensraum liegt dabei allein in der Zeichenfolge, die dadurch bezeichnete Ressource muss nicht existieren. Es können auch mehrere Namensräume in einem XML-Dokument verwendet werden. Dann werden den Namensräumen Präfixe zugewiesen, die den Elementnamen zur Kennzeichnung mit einem Doppelpunkt vorangestellt werden, bspw. so:

```
<x4t:element xmlns:x4t="http:\x4t.uni-muenster.de"/>
```

Der qualifizierte Name (QName) des Knotens besteht dann aus dem Präfix (Prefix) und der lokalen Bezeichnung (LocalPart).

Mittels der Kommentarzeichen `<!--` und `-->` kann Text eingeschlossen werden, der durch die XML-verarbeitende Anwendung nicht interpretiert wird.

Der Vollständigkeit halber seien noch Processing Instructions als Konstrukt von XML genannt, mit denen XML-verarbeitenden Programmen Anweisungen mitgeteilt werden. Die Syntax folgt dabei der Form:

```
<?ANWENDUNG Beliebige Anweisung?>
```

Wenn ein XML-Dokument durch die Verwendung von Processing Instructions Anweisungen für eine bestimmte Zielanwendung enthält, verliert XML die Funktion der reinen Datenauszeichnung und büßt durch anwendungsspezifische Bestandteile Interoperabilität ein. Daher wird das Konzept der Processing Instructions ambivalent gesehen [Gol91] und in der vorliegenden Arbeit nicht verwendet.

Programm 3.1 zeigt ein XML-Dokument, das Informationen über eine Person in strukturierter Weise enthält. Die Struktur folgt dabei der Struktur des ODM, einem Datenmodell für klinische Studien, das in Abschnitt 3.4.4 noch näher beschrieben wird, und verwendet dessen Namensraum. Zeile 1 enthält die XML-Deklaration, die die XML-Version und die Zeichencodierung der Datei angibt. Darauf folgend steht in Zeile 2 das Wurzelelement `AdminData`, das in diesem einfachen Fall nur die Namensraumdeklaration und einen Kindknoten

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AdminData xmlns="http://www.cdisc.org/ns/odm/v1.3">
3    <!-- This is the section for users data -->
4        <User OID="forster" UserType="Other">
5          <FirstName>Christian</FirstName>
6          <LastName>Forster</LastName>
7          <Email>christian.forster@ercis.de</Email>
8        </User>
9  </AdminData>
```

Programm 3.1: Ein XML-Dokument, das Informationen über eine Person enthält.

`User` enthält. Dieses Element enthält die beiden Attribute `OID` und `UserType`, denen entsprechende Werte zugeordnet sind. Die Kindelemente in den Zeilen 5 bis 7 enthalten jeweils Text. Die folgenden Zeilen schließen die geöffneten Elemente wieder. Das gezeigte XML-Dokument ist wohlgeformt, weil es die syntaktischen Regeln einhält, gibt aber keine Auskunft darüber, ob es einem Datenschema folgt und somit auf Gültigkeit geprüft werden kann, womit sich der folgende Abschnitt beschäftigt.

3.1.3 Validierung von Dokumenten

Neben der Wohlgeformtheit, die jedes XML-Dokument aufweisen muss, um mit Techniken der XML-Sprache verarbeitet zu werden, lassen sich Spezifizierungen vornehmen, um die Gültigkeit eines Dokuments in einem Verwendungskontext zu validieren. Durch die XML Schemabeschreibung können für einen Anwendungszweck *gültige* XML-Dokumente als Untermenge der wohlgeformten XML-Dokumente bestimmt werden. Dabei kommen unterschiedliche Sprachen zur Schemabeschreibung zum Einsatz, bspw. Document Type Definition (DTD), Relax NG und W3C XML Schema Definition (XSD) [37]. Diese unterscheiden sich in Syntax, Mächtigkeit und Einsatzgebiet. Der im Rahmen des Anwendungsbeispiels verwendete und in Abschnitt 3.4.4 besprochene Standard CDISC ODM liegt, wie viele andere Schemadefinitionen auch, als W3C XSD vor. Somit konzentriert sich die Darstellung hier auf diese Schemasprache. Deren Spezifikation wird durch das W3C vorgenommen und umfasst die beiden Bestandteile Struktur [GST12] und Datentypen [PGM+12]. Detaillierte Ausführungen finden sich in der Literatur sowohl zur etablierten Version 1.0 [Vli03] als auch zur Neufassung 1.1 [Wal12], deren derzeit aktueller Stand aus dem Jahr

2012 stammt.

Schemadefinitionen in XSD sind selbst XML-Dokumente des Namensraumes `http://www.w3.org/2001/XMLSchema`, der meist mit dem Präfix `xs` referenziert wird. Ein Schema bestimmt die im Instanzdokument zulässigen XML-Sprachkonstrukte, indem ihre Strukturierung als Baum und ihr Inhalt definiert werden. Die Sprache bietet mächtige Konstrukte, mit denen die Strukturierung, die Wiederverwendbarkeit und die Erweiterbarkeit einer Schemadefinition beeinflusst werden können. Es gibt daher häufig viele Arten, ein Zielschema zu beschreiben. Um die Funktionsweise beispielhaft deutlich zu machen, wird im Folgenden ein vereinfachter Ausschnitt der Schemadefinition des ODM erläutert, die dem Dokument aus Programm 3.1 zu Grunde liegt. Darin sind die Definition und Verwendung von Elementen und Attributen zu finden. Die dabei verwendeten Datentypen lassen sich grundlegend in einfache und komplexe Datentypen unterscheiden. Einfache Datentypen werden von Attributen und Elementen verwendet. Sie sind Basistypen oder basieren auf der Ableitung von Basistypen. Basistypen sind vordefinierte Typen wie `xs:string`, `xs:integer`, `xs:boolean` und `xs:time`. Diese können durch Einschränkung, Auflistung und Vereinigung zu neuen einfachen Datentypen abgeleitet werden.

Komplexe Datentypen zeichnen sich dadurch aus, dass sie Kindelemente enthalten und Attribute aufweisen können. Je nach Inhaltsmodell werden komplexe Datentypen in solche mit einfachem und komplexem Inhaltsmodell unterschieden. Ein einfaches Inhaltsmodell besteht aus einem einfachen Datentypen. Ein Element komplexen Typs mit einfachem Inhaltsmodell unterscheidet sich von einem Element einfachen Typs dadurch, dass es über Attribute verfügt. Ein komplexes Inhaltsmodell enthält eine Auflistung der zulässigen Elemente und ggf. Attribute. Ein Element mit komplexem Inhaltsmodell ist stets ein Element komplexen Typs.

In Programm 3.2 wird das Schema wie folgt aufgebaut. Zeile 1 öffnet das Wurzelelement `xs:schema`, gibt die verwendeten Namensräume an und informiert den Schema-Prozessor durch Angabe des Attributs `targetNamespace` über den Namensraum, der durch das Schema beschrieben wird. Sodann folgen in den Zeilen 2 bis 14 die Definitionen der einfachen Datentypen `oid` und `UserType`. Der Typ `oid` wird dabei auf Basis des Datentyps `xs:string` definiert, mit der Einschränkung, dass die minimale Länge 1 ist. Für den Typ `UserType` wird derselbe Basistyp verwendet, allerdings erfolgt die Beschränkung der gültigen Werte hierbei durch Auflistung des Elements `xs:enumeration`. In den Zeilen 15 bis 17 werden einfache Elemente, die Inhalt vom Typ `xs:string` enthalten, definiert. Darauf folgend wird in den Zeilen 18 bis 26der komplexe Daten-

typ ODMcomplexTypeDefinition-User gebildet. Er verwendet die drei zuvor definierten Elemente, wobei durch den Wert 0 des Attributes minOccurs bestimmt wird, dass sie optional sind und zusätzlich im Fall von Email durch das Attribut maxOccurs mit dem Wert unbounded festgelegt wird, dass es beliebig oft auftreten darf. Des Weiteren verwendet der Datentyp die beiden zuvor definierten einfachen Datentypen oid und UserType als Attribute, wobei das Attribute OID wegen seiner Eigenschaft use="required" auftreten muss, während UserType optional ist. Der so definierte komplexe Datentyp wird in Zeile 27 für das Element User verwendet. Dieses Element wird darauf folgend als Inhalt des komplexen Datentyps ODMcomplexTypeDefinition-AdminData verwendet, der analog dazu definiert und im Element AdminData verwendet wird. Die Definition des Elements AdminData weist in Zeile 34 das Element xs:unique auf. Dieses enthält eine Restriktion in Bezug auf die erlaubten Inhalte, die sicherstellt, dass die enthaltenen Elemente User über ein eindeutiges Schlüsselattribut OID verfügen.

```
1  <xs:schema xmlns="http://www.cdisc.org/ns/odm/v1.3"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.cdisc.org/ns/odm/v1.3" >
2    <xs:simpleType name="oid">
3      <xs:restriction base="xs:string">
4        <xs:minLength value="1"/>
5      </xs:restriction>
6    </xs:simpleType>
7    <xs:simpleType name="UserType">
8      <xs:restriction base="xs:string">
9        <xs:enumeration value="Sponsor"/>
10       <xs:enumeration value="Investigator"/>
11       <xs:enumeration value="Lab"/>
12       <xs:enumeration value="Other"/>
13     </xs:restriction>
14   </xs:simpleType>
15   <xs:element name="FirstName" type="xs:string"/>
16   <xs:element name="LastName" type="xs:string"/>
17   <xs:element name="Email" type="xs:string"/>
18   <xs:complexType name="ODMcomplexTypeDefinition-User">
19     <xs:sequence>
20       <xs:element ref="FirstName" minOccurs="0"/>
21       <xs:element ref="LastName" minOccurs="0"/>
22       <xs:element ref="Email" minOccurs="0" maxOccurs="
             unbounded"/>
```

```
23      </xs:sequence>
24      <xs:attribute name="OID" type="oid" use="required"/>
25      <xs:attribute name="UserType" type="UserType"/>
26    </xs:complexType>
27    <xs:element name="User" type="ODMcomplexTypeDefinition-
          User"/>
28    <xs:complexType name="ODMcomplexTypeDefinition-
          AdminData">
29      <xs:sequence>
30        <xs:element ref="User" minOccurs="0" maxOccurs="
              unbounded"/>
31      </xs:sequence>
32    </xs:complexType>
33    <xs:element name="AdminData" type="
          ODMcomplexTypeDefinition-AdminData">
34      <xs:unique name="UC-AD-1">
35        <xs:selector xpath="User"/>
36        <xs:field xpath="@OID"/>
37      </xs:unique>
38    </xs:element>
39  </xs:schema>
```

Programm 3.2: Ausschnitt der Schemadefinition des ODM.

Mit dem im Beispiel gezeigten XML Schema kann das XML-Dokument 3.1 validiert werden. Das Schema entstammt der ODM-Spezifikation und wurde zur besseren Darstellbarkeit bereits vereinfacht. Mittels XSD-Schemakonstrukten hätte noch eine über die ODM-Spezifikation hinausgehende, sinnvolle weitere Einschränkung vorgenommen werden können. Für das Element `Email`, das die E-Mail-Adresse der Person bezeichnet, wird lediglich der Datentyp `xs:string` gefordert, während die Spezifikation für E-Mail-Adressen RFC 5322 [Res08] ein strengeres Muster vorschreibt. So muss die Zeichenkette einen lokalen Teil aufweisen, der durch das Zeichen @ vom Domänenteil getrennt wird, was durch einen regulären Ausdruck [Stu07] folgender Art spezifiziert wird: `<xs:pattern value="[^@]+@[^@]+"/>`[2].

XML Schema-Dateien sind versionierbar, so dass durch Weiterentwicklung hervorgebrachte Versionsstände identifizierbar sind. Das Publizieren neuer Versionen wird als *Schemaevolution* bezeichnet und kann zu inkompatiblen Versio-

[2] Es sei angemerkt, dass diese einfache Form hier als Beispiel für die Verwendung regulärer Ausdrücke angeführt ist, aber nicht sämtliche ungültigen E-Mail-Adressen verhindert. Der vollständige reguläre Ausdruck für E-Mail-Adressen ist deutlich komplizierter [20].

nen führen [GLQ11]. In praxi wird bei der Entwicklung neuer Versionen aus Gründen der Interoperabilität auf Kompatibilität geachtet, so dass bestehende Dokumente auch mit der weiterentwickelten Version des XML Schemas validiert werden können. Diese Kompatibilität wird nur aufgegeben, wenn neue Anforderungen unvereinbar mit dem bestehenden Schema sind. Kompatible Änderungen sind Erweiterungen um optionale Konstrukte, Neu-Interpretierung bestehender Konstrukte unter Beibehaltung der Dokumentstruktur und Umstrukturierung des Schemas im Rahmen von Refactoring [MML07].

3.1.4 Modellieren von XML

Der Erstellung von Spezifikationsdokumenten in XML Schema oder ggf. einer anderen Beschreibungssprache liegt ein mindestens implizit beim Ersteller gedachtes Modell über die Beschaffenheit gültiger Dokumente zu Grunde. Für eine saubere Dokumentation ist das Explizieren dieses Modells wünschenswert, allerdings existiert keine allgemein anerkannte Modellierungstechnik.

Lehrbücher zu XML Schema behandeln das Thema Modellierung nicht ausführlich. So findet das Thema in [Vli03] keine Berücksichtigung. In [SWW11a] werden sehr einfache Block- und Baumdiagramm-Darstellungen zur Modellierung vorgestellt, dann aber nicht weiter verwendet.

Eine Reihe wissenschaftlicher Publikationen stellt Erweiterungen von ERM und UML um Konzepte, die für die XML Schema-Modellierung benötigt werden, vor.

Zu den ERM-basierten Ansätzen gehören: *XER* [SMD03], *X-Entity* [LSR03] und *XSEM* [Nec07].

Mit dem Ansatz XER [SMD03] wird ein Modell erzeugt, das sich direkt in ein XML Schema umsetzen lässt. Dazu wird ERM um Entitätstypen erweitert, die Attribute genannte, untergeordnete Entitätstypen geordnet und ungeordnet aufnehmen können. Ebenfalls ist in der Modellierung eine Notation für mixed content, der aus Text und Elementen besteht, vorgesehen. Für IS-A Beziehungen wird eine vom ERM-Standard abweichende Syntax verwendet. Relationstypen der Kardinalität n:m sind in diesem Modell nicht vorgesehen.

Der Ansatz X-Entity [LSR03] zielt auf das Erzeugen von erweiterten ERM Modellen aus XML Schemata. Die dabei verwendeten Entitätstypen sind um Modellelemente für optionale sowie disjunkte Attribute erweitert. Für 1:n Relationstypen wird eine von ERM abweichende Notation mit gerichteten Kanten verwendet. Die Autoren geben Transformationsregeln für die Erzeugung des Modells aus einem XML Schema an, die umgekehrte, im Zuge der Softwareentwicklung benötigte Erzeugung eines XML Schemas aus einem Modell ist nicht

beschrieben, erscheint aber möglich. Die Modellierung von n:m Relationstypen, mixed content und der Reihenfolge von Elementen ist in X-Entity nicht möglich.

Beide Ansätze erfordern Erweiterungen der grafischen ERM Notation und werden nicht durch derzeit erhältliche Tools unterstützt.

Das Vorgehen XSEM setzt auf eine zweistufige Modellierung, bei der während der Analyse ein konzeptionelles Modell, genannt XSEM-ER, und während des Entwurfs ein logisches Modell, genannt XSEM-H, erstellt wird. XSEM-ER modifiziert ERM hinsichtlich des Identitätskonzepts für Entities, das nicht mehr auf einer Gleichheit der Attributwerte basiert, sondern auch unterschiedliche Entities mit gleichen Attributen zulässt, die sich durch die Reihenfolge unterscheiden lassen. Hinzu kommen die Konzepte „Outgoing Cluster Type“ und „Incoming Cluster Type“, mit denen sich semistrukturierter Inhalt und mixed content darstellen lassen. Das XSEM-H Modell ist eine hierarchische Darstellung des XSEM-ER Modells, das keine zusätzliche Semantik enthält, und durch die Anwendung von Transformationsregeln erzeugt wird. Der XSEM-Ansatz beschränkt sich auf die Beschreibung von XML Dokumentstrukturen als XSEM-ER und XSEM-H Modelle, die Erzeugung einer Schemabeschreibung ist nicht Bestandteil der Methode.

Ansätze auf Basis von UML weisen den Vorteil auf, dass UML über einen *Profile* genannten Erweiterungsmechanismus verfügt. Auf Klassendiagrammen basieren *GXSL* [LO01], eine unbenannte *Erweiterung* von UML [RBG02], *XUML* [LLY06] und ein *UML Profil* für XML [BKK03].

Graphical XML Schema Definition Language (GXSL) [LO01] erweitert UML um neue Notationselemente und wurde im Zusammenhang mit XML-Netzen vorgestellt. So werden ein neues Icon für textlichen Inhalt und Markierungen für leere Elemente und für solche mit beliebigem Inhalt eingeführt. Um Inhalte von Elementen näher zu spezifizieren, wird eine Notation für Reihenfolgen von Elementen eingeführt. Eine Notation für Elemente, die nur alternativ auftreten dürfen, ist ebenfalls vorhanden. Schließlich ist es möglich, Integritätsbedingungen für Fremdschlüssel abzubilden. GXSL basiert in der vorgestellten Variante auf DTD, so dass Funktionen für das mächtigere XML Schema fehlen.

Der Ansatz aus [RBG02] enthält die Beschreibung eines dreistufigen Verfahrens, das für die Analysephase ein konzeptionelles Modell enthält, aus dem in der Entwurfsphase ein logisches Modell abgeleitet wird, das schließlich zur Implementierung des Schemas führt. Das konzeptionelle Modell wird als Klassendiagramm erzeugt, wobei eine Erweiterung zur Angabe des Primärschlüssels vorgesehen ist. Die Ableitung des logischen Schemamodells geschieht anhand von konfigurierbaren Regeln, die die Umsetzung von Datentypen und das Auflö-

sen der Assoziationen in komplexe, geschachtelte Datentypen angeben. Das so erzeugte logische Modell ist eine eindeutige Spezifizierung des XML Schemas, das daraus generierbar ist.

Das XUML [LLY06] genannte Verfahren setzt ebenfalls auf eine konzeptionelle und logische Modellebene, die konzeptionelle Ebene wird in UML Klassendiagrammen entwickelt, während für das logische Modell eine Erweiterung von XML erfolgt. Diese führt eine „generic Aggregation" genannte Assoziationsart ein. Sie hat die Bedeutung einer n-ären Komposition, bei der n Klassen nur gleichzeitig Teil der Kompositionsklasse sind und soll zur Abbildung hierarchischer Strukturen dienen. Eine alternative Darstellung, die UML Kompositionsstrukturdiagramme verwendet, wird ebenfalls präsentiert. Der Transformationsschritt vom logischen Modell zum XML Schema wird in der Beschreibung von XUML nicht erläutert.

Das UML Profil [BKK03] bietet den umfassendsten Ansatz [BKK04] zur Modellierung von XML Schema mit UML. Es zielt darauf ab, beliebige XML Schemata mit Mitteln von UML abzubilden. Dazu stellt es eine Reihe von Stereotypen bereit, die UML Klassendiagramme erweitern. Aus mit diesen Sprachelementen aufgebauten Modellen lassen sich dann XML Schemata ableiten. Leider findet sich zu dem im Ausblick der Arbeit angekündigten Prototypen keine Publikation, so dass keine Softwareunterstützung der Methode angenommen werden kann.

Die Autoren der Anwendung XCASE [KKLM10] versprechen eine Toolunterstützung zur Modellierung nach dem XSEM-Modell, die Anwendung setzt aber ein auf UML Klassendiagrammen basierendes Modell ein. Es basiert auf dem mehrstufigen Ansatz, zunächst ein plattformunabhängiges Modell zu erzeugen, das zur Analysephase der Softwareentwicklung passt, und daraus ein plattformspezifisches Modell abzuleiten, das zur Entwurfsphase passt. Schließlich kann die Anwendung aus dem plattformspezifischen Modell ein XML Schema generieren. Die Modellierungssoftware ist unter der Lizenz GNU General Public License (GPL) frei erhältlich[3], allerdings wurde die letzte Aktualisierung im Jahr 2010 veröffentlicht, so dass die zukünftige Perspektive unklar erscheint.

Ein Ansatz, der für die Abbildung von Geschäftsobjekten entwickelt wurde und auf Asset Oriented Modelling (AOM) [Dau03] und ERM basiert, findet sich in [SVOK11]. Grundlage ist die hierarchische Darstellung von Geschäftsobjekten mittels AOM. Beziehungs- und Vererbungsabhängigkeiten zwischen diesen werden direkt durch Kanten modelliert. Beziehungskanten entsprechen der Semantik von ERM, es werden aber keine Relationstypen verwendet. Mehrere Be-

[3] http://xcase.codeplex.com/

ziehungskanten an einem Objekt können mit Sammelbedingungen als logische Verknüpfungen *Und*, *Oder* und *exklusives Oder* weiter spezifiziert werden. Vererbungskanten werden durch eine auf die Generalisierung gerichtete Kante dargestellt. Der Ansatz ist in der Software Horus Business Modeler[4] implementiert und in der Version Horus Freeware kostenlos erhältlich. Mittels dieser lassen sich XML Schemata aus den Modellen generieren.

Wichtig für die Wahl der Modellierungstechnik im Rahmen der Softwareentwicklung sind neben der theoretischen Angemessenheit der Methode auch die Verbreitung und die Verfügbarkeit von Modellierungstools. Für die Entwicklung von Software als XML-basierte Architektur ist wichtig, dass das erzeugte Modell im Verlauf des Entwicklungsprozesses erzeugt und bearbeitet werden kann und anschließend als Dokumentation zur Verfügung steht. Welche Technik dafür zum Einsatz kommt, ist nicht von primärer Bedeutung.

3.2 XQuery

Der vorherige Abschnitt hat sich mit dem Aufbau und der Validierung von XML-Dokumenten und der Modellierung ihrer Spezifikation beschäftigt. Im Folgenden wird mit XQuery die Programmiersprache vorgestellt, mit der Abfragen auf XML-Dokumenten nativ durchgeführt werden können. Dazu werden der grundlegende Aufbau von XQuery sowie die Navigation mit XPath in einem XML-Dokument gezeigt und FLWOR-Ausdrücke als für XQuery elementare Konstrukte erläutert. Darüber hinaus werden die Möglichkeiten von XQuery, die die reine Abfrage von XML-Dokumenten übersteigen, präsentiert.

3.2.1 Grundlegender Aufbau

Die XQuery-Sprache basiert auf der Spezifikation verwandter Sprachstandards. Für die derzeit den Spezifikationsprozess durchlaufende Version XQuery 3.0 [RCDS13b] sind dies:

- XPath 3.0 [RCDS13a]
- XQuery and XPath Data Model (XDM) 3.0 [WBS13]
- XML Schema 1.0 oder 1.1 [GST12; PGM+12]
- XQuery and XPath Functions and Operators 3.0 [Kay13]

[4] http://www.horus.biz/

- XQueryX 3.0 [Mel13]

XPath dient der Navigation in XML-Dokumenten und ist eine Teilmenge der XQuery-Spezifikation. XQuery and XPath Data Model (XDM) ist das Datenmodell, das die durch XQuery-Ausdrücke verarbeitbaren Werte festlegt. Die Datentypen entstammen der Spezifikation XML Schema, wobei die Implementierung sowohl für Version 1.0 als auch 1.1 erlaubt ist. Die Spezifikation XQuery and XPath Functions and Operators legt fest, welche Funktionen und Operatoren die Implementierung bereitstellen muss. XQueryX ist eine alternative Syntax für XQuery, die auf XML basiert. Sie findet im Rahmen dieser Arbeit keine Verwendung.

Die durch das W3C verabschiedete und derzeit aktuelle Version XQuery 1.0 [BCF+10] stammt aus dem Jahr 2010. Sie basiert auf XPath Version 2.0 [BBC+10]. Der Spezifizierungsprozess [Jac05] für die Nachfolger in Version 3.0 befindet sich derzeit in der Phase „Candidate Recommendation", so dass zwar noch Anpassungen im Detail, aber keine signifikanten Änderungen mehr an der Spezifikation zu erwarten sind [Eis13]. Aufgrund interessanter neuer Bestandteile wird in dieser Arbeit von Version 3.0 Gebrauch gemacht. Wenn im Rahmen dieser Arbeit Konstrukte verwendet werden, die erst in den jeweiligen Versionen 3.0 spezifiziert sind, wird darauf hingewiesen. Derzeit zu XQuery erhältliche einführende Literatur behandelt die Version 1.0 [LS04; MB06].

Das Sprachkonzept von XQuery lehnt sich an das funktionale Programmierparadigma [RL99; CK02] an. Diese Art von Sprachen orientiert sich am mathematischen Funktionsbegriff, bei dem Elemente einer Definitionsmenge Elementen einer Wertemenge durch Funktionen zugeordnet werden. Im Vordergrund steht der Wert eines Ausdrucks, der bestimmt werden kann, sobald die Parameter des Funktionsaufrufs bekannt sind. Die Berechnung des Wertes hängt nicht von sonstigen Zuständen ab und hat keine Seiteneffekte. Daher lassen sich komplexe Ausdrücke durch Komposition aus einfacheren zusammensetzen und schrittweise durch Bestimmung der jeweiligen Eingabewerte berechnen. Variablen, deren Wert in imperativen Sprachen nach der Initialisierung änderbar ist, gibt es in funktionalen Sprachen daher nicht. Das Konzept der referentiellen Transparenz sieht vielmehr vor, dass Ausdrücke zu jeder Zeit denselben Wert enthalten. Daher ist die Reihenfolge der Berechnung auch nicht relevant. Aufgrund dieser Eigenschaften kann eine rein funktionale Sprache keine Eingaben und Ausgaben handhaben, da diese vom Systemzustand abhängen bzw. diesen verändern. XQuery ist allerdings nicht als rein funktionale Sprache spezifiziert, zudem weisen die unterschiedlichen Implementierungen Erweiterungen,

die mit Seiteneffekten behaftet sind, wie Schreiboperationen, auf[5]. Das Konzept des Ausdrucks ist allerdings auch für XQuery zentral: Konstituierend für XQuery sind XPath- und FLWOR-Ausdrücke.

3.2.2 XPath

Hauptanwendungszweck von XPath-Ausdrücken ist die Selektion von Knoten in XML-Dokumenten. Es können damit sowohl einzelne als auch mehrere Knoten ausgewählt werden. XPath ist eine typisierte Sprache, die die Datentypen des XDM Datenmodells verwendet. Dies verwendet die geordnete, nicht-geschachtelte Liste (sequence) von beliebig vielen Einträgen (items) als Grundkonstrukt. Die leere Liste wird dabei durch ein leeres Klammerpaar `()` gekennzeichnet, eine Liste mit einem einzigen Eintrag `(a)` ist identisch mit diesem Eintrag `a`, und der Kommaoperator separiert mehrere Einträge in einer Liste `(a,b,c)`. Die Spezifikation unterscheidet syntaktisch zwischen *Expr* und der Spezialisierung *ExprSingle* genannten Ausdrücken. Sie bestehen beide aus Listen, die durch Anwendung des Kommaoperators gebildet werden, wobei bei Expr-Ausdrücken die äußeren Klammern entfallen können, während sie bei ExprSingle-Ausdrücken notwendig sind. Diese Unterscheidung ist wichtig bei Ausdrücken, in denen das Komma eine andere Funktion, bspw. das Trennen der Argumente bei einem Funktionsaufruf, übernehmen kann.

Ein Eintrag ist dabei ein Knoten (node), ein atomarer Wert (atomic value) oder ab Version 3.0 auch eine Funktion (function). Zu den Knoten gehören die bereits in Abschnitt 3.1.2 beschriebenen Einheiten Dokument, Element, Attribut, Text, Namensraum, Processing Instruction und Kommentar. Atomare Werte sind Werte eines atomaren Typs, der zu einem der 21 in der Spezifikation festgelegten primitiven einfachen Typen gehört oder durch Einschränkung davon abgeleitet ist. Funktionen sind aufrufbare Einheiten, die in Abhängigkeit der übergebenen Parameter einen Rückgabewert liefern. Eine vertiefende Betrachtung des Funktionskonzepts geschieht im Rahmen von Abschnitt 3.2.4.

Die Navigationspfade von XPath orientieren sich entlang verschiedener Achsen (axes) der XML-Baumstruktur. Ein Pfadausdruck (path expression) kann aus beliebig vielen relativen Pfadausdrücken zusammengesetzt sein, die durch den Pfadoperator `/` voneinander getrennt sind. Die relativen Pfadausdrücke sind relativ zum jeweiligen Kontext und bestehen aus einem oder mehreren Schritten (steps). Ein Schritt generiert durch einen Achsenschritt (axis step) eine Ergebnis-

[5] Rein funktionale Programmiersprachen bieten hierzu das Konzept der Monaden an, das aber in XQuery keine Verwendung findet.

Tabelle 3.1: In XPath verfügbare Navigationsachsen.

Achse	Bedeutung
Vorwärtsachsen	
`child::`	Direkt untergeordnete Knoten. Dies ist die Standardachse, wenn keine Achsenbezeichnung angegeben ist
`descendant::`	Alle untergeordneten Elemente. Abkürzung mit `//`
`attribute::`	Die Attribute des Kontextknotens. Abkürzung mit `@`
`self::`	Der Kontextknoten selbst. Abkürzung mit `.`
`descendant-or-self::`	Vereinigung aus `descendant::` und `self::`
`following-sibling::`	Nachfolgende Knoten mit demselben übergeordneten Knoten
`following::`	Alle nachfolgenden Knoten
Rückwärtsachsen	
`parent::`	Der direkt übergeordnete Knoten
`ancestor::`	Alle übergeordneten Knoten
`preceding-sibling::`	Vorherige Knoten mit demselben übergeordneten Knoten
`preceding::`	Alle vorherigen Knoten
`ancestor-or-self::`	Vereinigung aus `ancestor::` und `self::`

sequenz und wendet darauf ggf. Prädikate zur Filterung an. Der Achsenschritt bestimmt durch die Angabe der Achse gleichsam die Richtung, in der die Ergebnissequenz durch Knotentests gesucht werden soll. Die Achsen lassen sich in Vorwärtsachsen (forward axes) für untergeordnete und nachfolgende Knoten und Rückwärtsachsen (reverse axes) für übergeordnete und vorherige Knoten einteilen. Es stehen die in Tabelle 3.1 genannten Achsen zur Verfügung.

Für häufig verwendete Achsen gibt es abkürzende Schreibweisen. Die Achse `child::` wird als Standard implizit angenommen, wenn keine Bezeichnung angegeben ist. Für die Achse `descendant::` steht der doppelte Schrägstrich zur Verfügung und für die Achse `attribute::` das Zeichen `@`.

Mit Knotentests können der Typ des Knotens und der Name geprüft werden. Prädikate sind Ausdrücke mit booleschem Wertebereich und werden für jeden durch den Knotentest identifizierten Knoten ausgewertet. Hiermit werden häufig Tests des jeweiligen Knoteninhalts durchgeführt. Prädikate stehen in

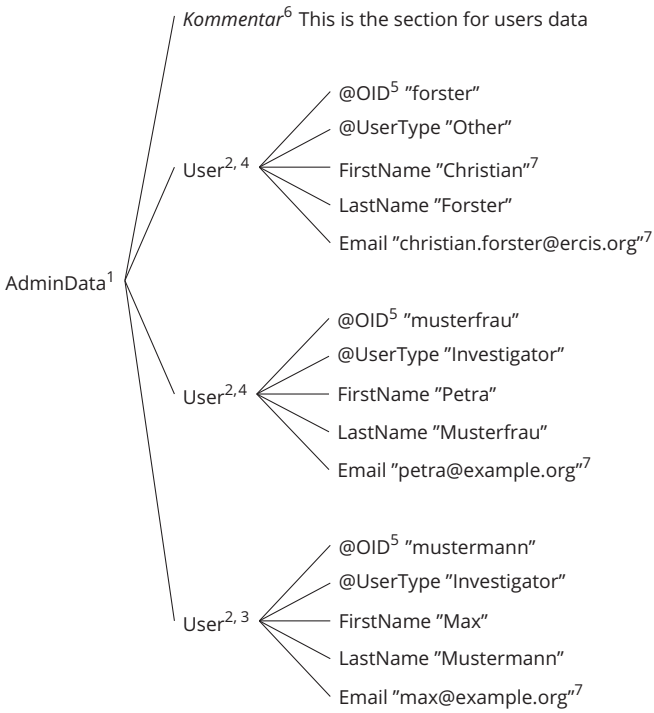


Abbildung 3.1: Visualisierung des Dokuments aus Programm 3.3. Die hochgestellten Zahlen an den Elementen zeigen, dass der Knoten durch den XPath-Ausdruck in der entsprechenden Zeile aus Programm 3.4 ausgewählt wird.

eckigen Klammern hinter dem Knotentest. Diese Struktur der XPath-Ausdrücke soll anhand der Beispiele in Programm 3.4 illustriert werden. Für die Anwendung der Ausdrücke wird eine erweiterte Variante, zu finden in Programm 3.3, des aus Programm 3.1 bekannten XML-Dokuments verwendet. Die Erweiterung besteht im Hinzufügen weiterer User-Knoten und ist gültig im Sinne des in Programm 3.2 gezeigten XSD Schemas. Abbildung 3.1 zeigt zur Verdeutlichung eine Visualisierung des Dokuments als Baum.

Das Ergebnis des in Zeile 1 von Programm 3.4 gezeigten Ausdrucks ist eine Liste mit dem Inhalt `odm:AdminData`. Er wird selektiert, indem zuerst als

```
<?xml version="1.0" encoding="UTF-8"?>
<AdminData xmlns="http://www.cdisc.org/ns/odm/v1.3">
  <!-- This is the section for users data -->
  <User OID="forster" UserType="Other">
    <FirstName>Christian</FirstName>
    <LastName>Forster</LastName>
    <Email>christian.forster@ercis.de</Email>
  </User>
  <User OID="musterfrau" UserType="Investigator">
    <FirstName>Petra</FirstName>
    <LastName>Musterfrau</LastName>
    <Email>petra@example.org</Email>
  </User>
  <User OID="mustermann" UserType="Investigator">
    <FirstName>Max</FirstName>
    <LastName>Mustermann</LastName>
    <Email>max@example.org</Email>
  </User>
</AdminData>
```

Programm 3.3: Ein XML-Dokument, das Informationen über mehrere Personen enthält und unter dem Dateinamen `admindata.xml` gespeichert ist.

```
doc("admindata.xml")/odm:AdminData
doc("admindata.xml")/odm:AdminData/odm:User
doc("admindata.xml")/odm:AdminData/odm:User[@OID="
    mustermann"]
doc("admindata.xml")/odm:AdminData/odm:User[@OID="
    mustermann"]/preceding-sibling::odm:User
doc("admindata.xml")/odm:AdminData/odm:User/@OID
doc("admindata.xml")/odm:AdminData/comment()
doc("admindata.xml")//(odm:FirstName | odm:Email)/string
    (.)[string-length(.) >8]
```

Programm 3.4: XPath-Ausdrücke. Das Präfix `odm` sei extern an den entsprechenden Namensraum gebunden.

Kontextknoten die Datei admindata.xml ausgewählt wird. Daraus wird auf der Kindachse per Knotentest der Knoten mit dem qualifizierten Namen odm:AdminData ausgewählt. Da keine Prädikate oder weitere relative Pfadausdrücke folgen, wird der Knoten zurückgegeben. Der nächste Ausdruck in Zeile 2 ist mit dem Pfadoperator um die Selektion der Knoten odm:User erweitert. Ausgehend von der vorherigen Selektion werden hierbei die drei Elemente mit dem qualifizierten Namen odm:User gewählt und zurückgegeben. Der folgende Ausdruck in Zeile 3 verwendet ein Prädikat. Nur solche Knoten werden selektiert, bei denen das mit @OID referenzierte Attribut den Wert mustermann aufweist. Es wird also nur das Element User aus Zeile 14 zurückgegeben. Der Ausdruck in Zeile 4 baut darauf auf, verwendet aber nicht ausschließlich die Kindachse. Die Achse preceding-sibling:: wählt alle vorhergehenden Geschwisterelemente, also solche mit dem gleichen übergeordneten Element, aus. Bezogen auf das Beispiel sind die Elemente User in den Zeilen 4 und 9. Neben Elementen können auch sonstige Knoten selektiert werden, wie die Attribute OID in Zeile 5 oder der Kommentarknoten mit dem Ausdruck in Zeile 6. Der Ausdruck in Zeile 7 zeigt die Verwendung von Funktionen und Filterausdrücken. Zunächst wird die Achse descendant:: in abgekürzter Schreibweise // gewählt, sodann alle Elemente darunter, die den Namen odm:FirstName oder odm:Email enthalten. Auf diese wird die Funktion string() angewendet, die die Zeichenkettenrepräsentation des übergebenen Arguments zurückgibt. Diese Zeichenketten werden durch das folgende Prädikat weiter eingeschränkt, indem nur solche selektiert werden, die aus mehr als 8 Zeichen bestehen, bestimmt durch die Funktion string-length(.). Hier lautet die Ergebnissequenz:

```
("Christian", "christian.forster@ercis.de",
    "petra@example.org", "max@example.org")
```

Neben der Navigation in XML-Dokumenten mit homogener Struktur, wie in Abbildung 3.1 abgebildet, erlaubt XPath die Navigation in XML-Dokumenten mit unregelmäßigerem Aufbau. Programm 3.5 zeigt in Abwandlung des XML-Dokumentes Programm 3.3 Adressdaten, die weniger regelmäßig strukturiert sind. Durch die abweichende Struktur ist das Dokument nicht valide im Sinne des ODM-Schemas. So gibt es ausgelassene und mehrfach auftretende Elemente sowie Anordnungen der Elemente auf unterschiedlichen Hierarchieebenen.

Abbildung 3.2 zeigt zur Verdeutlichung eine Visualisierung des Dokuments als Baum. Das erste Element User enthält alle vormals als Elemente vorhandenen Daten in Form von Attributen. Das zweite Element User enthält nun zwei Elemente FirstName, die die Existenz mehrerer Vornamen repräsentieren. Das dritte Element User weist ein neues Kindelement Communication auf, um die

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AdminData xmlns="http://www.cdisc.org/ns/odm/v1.3">
3    <User OID="forster" UserType="Other" FirstName="
         Christian" LastName="Forster" Email="christian.
         forster@ercis.de"/>
4    <User OID="musterfrau" UserType="Investigator">
5      <FirstName>Petra</FirstName>
6      <FirstName>Maria</FirstName>
7      <LastName>Musterfrau</LastName>
8      <Email>petra@example.org</Email>
9    </User>
10   <User OID="mustermann" UserType="Investigator">
11     <FirstName>Max</FirstName>
12     <LastName>Mustermann</LastName>
13     <Communication>
14       <Email>max@example.org</Email>
15       <Email>mustermann@example.org</Email>
16       <Phone>555200</Phone>
17       <Pager>55523</Pager>
18       <Fax>55524</Fax>
19     </Communication>
20   </User>
21 </AdminData>
```

Programm 3.5: Ein XML-Dokument, das unregelmäßig strukturierte Informationen über mehrere Personen enthält und unter dem Dateinamen `userfile.xml` gespeichert ist.

unterschiedlichen Kommunikationsadressen der Person zu gruppieren.

Programm 3.6 zeigt die XPath-Ausdrücke, die diese Struktur berücksichtigen, und zugehörige Werte. Der Ausdruck in Zeile 1 selektiert die Werte der Elemente und Attribute, die `Email` heißen und an beliebiger Stelle im Dokument auftreten. Es werden alle im Dokument vorhandenen E-Mail-Adressen gesucht. Mit dem Ausdruck in Zeile 7 werden sämtliche Elemente ausgewählt, deren Inhalt numerisch interpretierbar ist. Angewendet auf das Beispieldokument sind dies die Telefon-, Pager- und Faxnummern. Der in Zeile 12 befindliche Ausdruck wählt das jeweils erste Element `Email`, das sich an beliebiger Position innerhalb eines Elementes `User` befindet. Der Ausdruck in Zeile 16 selektiert Attribute und Elemente mit dem Namen `FirstName` innerhalb des Elements

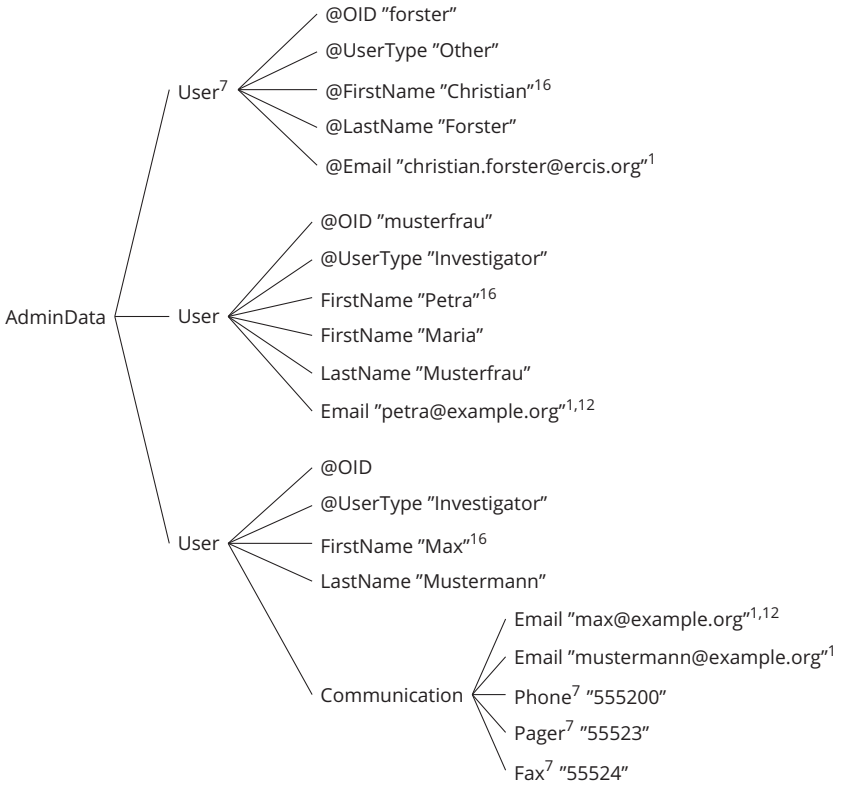


Abbildung 3.2: Visualisierung des Dokuments aus Programm 3.5. Die hochgestellten Zahlen an den Elementen zeigen, dass der Knoten durch den XPath-Ausdruck in der entsprechenden Zeile aus Programm 3.6 ausgewählt wird.

```
1  doc("/db/tmp/userfile.xml")//(./@Email | odm:Email)/
       string()
2  christian.forster@ercis.de
3  petra@example.org
4  max@example.org
5  mustermann@example.org
6  ==========
7  doc("/db/tmp/userfile.xml")//*[number(.) = number(.)]
8  <odm:Phone>555200</odm:Phone>
9  <odm:Pager>55523</odm:Pager>
10 <odm:Fax>55524</odm:Fax>
11 ==========
12 doc("/db/tmp/userfile.xml")/odm:AdminData/odm:User//
       odm:Email[1]
13 <odm:Email>petra@example.org</odm:Email>
14 <odm:Email>max@example.org</odm:Email>
15 ==========
16 doc("/db/tmp/userfile.xml")/odm:AdminData/odm:User/(@*|*)
       [name(.)="FirstName"][1]/string()
17 Christian
18 Petra
19 Max
```

Programm 3.6: XPath-Ausdrücke und deren Werte bei Anwendung auf das inhomogen strukturierte Dokument aus Programm 3.5. Das Präfix `odm` sei extern an den entsprechenden Namensraum gebunden.

`User` und gibt den Wert der jeweils ersten Treffer zurück. Dadurch werden alle ersten Vornamen ausgegeben.

Mittels XPath-Ausdrücken kann also auch in unregelmäßig strukturierten Dokumenten navigiert werden.

3.2.3 FLWOR

In Anlehnung an die aus dem relationalen Konzept bekannte SQL-basierte Selektion durch `SELECT` und `FROM` mit weiteren optionalen Bestandteilen wie `WHERE` und `ORDER BY` existiert in XQuery das Konzept des FLWOR-Ausdrucks. Das Akronym steht für die Bestandteile des Ausdrucks `for`, `let`, `where`, `order by` und `return`.

```
1  let $doc := doc("admindata.xml")
2  return $doc/odm:AdminData
```

Programm 3.7: Einfacher XQuery-Ausdruck, der das Dokument `admindata.xml` zurückgibt.

Ein FLWOR-Ausdruck fängt mit beliebig vielen `for`- und `let`-Anweisungen an, mit denen Variablen an die angegebenen Sequenzen gebunden werden. Diese Bindungen liefern jeweils einen Datenstrom an Tupeln (tuple stream)[6]. Die erste `for`- oder `let`-Anweisung in einem FLWOR-Ausdruck erzeugt den Datenstrom, der den folgenden als Eingabe dient. Diese fügen die jeweilige Bindung hinzu und geben den Datenstrom als Ausgabe weiter. Der Unterschied zwischen `let` und `for` ist, dass mittels `let` die gesamte Sequenz gebunden wird, während mittels `for` über die Einheiten der Sequenz iteriert wird und die nachfolgenden Bestandteile des Ausdrucks für jede Iteration einzeln ausgewertet werden. Optional folgt `where`, um den Eingabedatenstrom so zu filtern, dass die im Ausgabedatenstrom enthaltenen Einheiten den angegebenen Bedingungen genügen. Mittels des ebenfalls optionalen `order by` kann der Eingabedatenstrom sortiert ausgegeben werden. Der letzte Bestandteil des FLWOR-Ausdrucks ist die Anweisung `return`, mittels derer aus den Eingabedatenströmen das Ergebnis konstruiert wird.

Die Beispiele in Programm 3.7, 3.8 und 3.9 zeigen die Anwendung von FLWOR-Ausdrücken. Der Ausdruck in Zeile 1 bindet jeweils das aus Programm 3.3 bekannte Dokument an die Variable `$doc`.

Im Ausdruck in Programm 3.7 wird dessen Element `AdminData` sodann durch die Anweisung `return` zurückgegeben.

Das Beispiel in Programm 3.10 bedient sich der Anweisung `for`, mittels der eine Iteration über die Elemente `User` durchgeführt wird, die jeweils an die Variable `$user` gebunden werden. Mittels `order by` werden die Elemente nach dem Inhalt des Elements `FirstName` sortiert. Der Rückgabewert dieses Ausdrucks besteht aus einer Sequenz dieser drei geordneten `user` Elemente.

Der geschachtelte Ausdruck in Programm 3.9 erzeugt das in Programm 3.10 wiedergegebene Ergebnis, das die Nutzer je nach Wert des Attributs `UserType` in Gruppen einordnet. Zuerst wird dazu das Wurzelelement `RoleData` konstru-

[6] Die Bezeichnung tuple ist in der Spezifikation folgend definiert: „A tuple is a set of zero or more named variables, each of which is bound to a value that is an XDM instance." Damit ist ein Tupel in XQuery nicht mit dem Tupelbegriff des relationalen Konzepts zu verwechseln.

```
1  let $doc := doc(”admindata.xml”)
2  for $user in $doc/odm:AdminData/odm:User
3  order by $user/odm:FirstName
4  return $user
```

Programm 3.8: FLWOR-Ausdruck, der das Dokument `admindata.xml` aus Programm 3.3 liest und eine Sequenz der Elemente `User` in der Reihenfolge der Vornamen ausgibt.

iert. Der Inhalt dieses Elements wird wiederum durch einen FLWOR-Ausdruck konstruiert. Die geschweiften Klammern bewirken, dass der umschlossene Text als Ausdruck interpretiert wird. Darin werden die unterschiedlichen Werte des Attributs `UserType` mit der Anweisung `for` an die Variable `$role` gebunden und mit der folgenden Anweisung `order by` alphabetisch sortiert. Dann wird in der Anweisung `return` ein Element `Role` konstruiert, dessen Inhalt durch einen weiteren FLWOR-Ausdruck gebildet wird. Die Anweisung `for` iteriert wiederum über die Elemente `User`, wobei durch die Bedingung `where` nur solche des derzeit an `$role` gebundenen Typ zugelassen werden. Für die Rückgabe wird ein neues Element `User` konstruiert, für das das Attribut `OID` und sämtliche Kindelemente, selektiert durch den auf alle Elementnamen passenden Platzhalter `*`, übernommen werden.

```
1  let $doc :=  doc(”admindata.xml”)
2  return
3    <RoleData xmlns=”http://www.cdisc.org/ns/odm/v1.3”>
4    {
5    for $role in distinct-values($doc/odm:AdminData/
         odm:User/@UserType/string())
6    order by $role
7    return
8    <Role Type=”{$role}”>
9        {
10       for $user in $doc/odm:AdminData/odm:User
11       where $user/@UserType/string() = $role
12       return
13       <User>
14         {
15         ($user/@OID, $user/*)
16         }
17       </User>
```

```
18        }
19    </Role>
20    }
21  </RoleData>
```

Programm 3.9: Komplexerer FLWOR-Ausdruck, der das Dokument `admindata.xml` aus Programm 3.3 liest und die enthaltenen Elemente `User` nach Typ in einem Element `Role` gruppiert.

```
1  <RoleData xmlns="http://www.cdisc.org/ns/odm/v1.3">
2    <Role Type="Investigator">
3      <User OID="musterfrau">
4        <FirstName>Petra</FirstName>
5        <LastName>Musterfrau</LastName>
6        <Email>petra@example.org</Email>
7      </User>
8      <User OID="mustermann">
9        <FirstName>Max</FirstName>
10       <LastName>Mustermann</LastName>
11       <Email>max@example.org</Email>
12     </User>
13   </Role>
14   <Role Type="Other">
15     <User OID="forster">
16       <FirstName>Christian</FirstName>
17       <LastName>Forster</LastName>
18       <Email>christian.forster@ercis.de</Email>
19     </User>
20   </Role>
21 </RoleData>
```

Programm 3.10: Wert des FLWOR-Ausdrucks aus Programm 3.9.

FLWOR-Ausdrücke, wie die gezeigten Beispiele, bilden den Kern von Programmen, die in der Programmiersprache XQuery geschrieben sind. Ausstehende Aspekte werden im folgenden Abschnitt beschrieben.

3.2.4 Weitere Programmelemente

In XQuery formulierte, ausführbare Software wird gemäß der Spezifikation in Module gegliedert. Module beginnen mit der Deklaration der XQuery-Version und gliedern sich in Bibliotheken und Hauptmodule. Zweck der Bibliotheksmodule ist es, Funktionen bereitzustellen, die von anderen Modulen importiert

werden können, während Hauptmodule einen evaluierbaren Ausdruck enthalten. Beiden gemein ist der Prolog, in dem Importe, Funktionsdeklarationen und weitere Deklarationen stattfinden. Bibliotheksmodule enthalten außer dem Prolog nur eine diesem vorangestellte Moduldeklaration. Hauptmodule enthalten nach dem Prolog den Anfragerumpf (QueryBody). Dieser enthält den Ausdruck, der wiederum aus einer Komma-separierten Liste von einem oder mehreren Einzelausdrücken besteht. Neben den bereits gezeigten XPATH- und FLWOR-Ausdrücken können dies auch weitere Konstrukte, wie Fallunterscheidungen, quantifizierende Ausdrücke und, seit Version 3.0, auch Try-Catch-Ausdrücke zur Fehlerbehandlung sein. Diese Ausdrücke lassen sich stets zu einem im XDM Datenmodell definierten Wert evaluieren. Das Beispiel in Programm 3.11 zeigt ein Bibliotheksmodul, das vom Hauptmodul aus Programm 3.12 importiert wird. Das Hauptmodul enthält nun die Anweisung `import` in Zeile 2 um das Bibliotheksmodul zu laden. Im Anfragerumpf finden sich nur noch die Anweisungen, um das Dokument `admindata.xml` zu laden und der Funktion `role-nodes` zu übergeben. Das Bibliotheksmodul enthält die Moduldeklaration in Zeile 2, die die Angabe des Namensraumes umfasst. In Zeile 4 wird die Funktion `role-nodes` deklariert, als zulässiger Parameter wird Knoten erwartet, der Rückgabewert der Funktion ist vom Typ Element. Die Konvertierung erfolgt analog zu der aus Programm 3.9 bekannten Variante ohne das Bibliotheksmodul, so dass das Ergebnis des Aufrufs des Hauptmoduls identisch zu dem in Programm 3.10 gezeigten ist.

Das Beispiel illustriert den grundlegenden Aufbau eines XQuery-Programms. Für eine vollständige, formale Darstellung der zulässigen Syntax sei auf die Spezifikation und die dort aufgeführte Erweiterte Backus-Naur-Form (EBNF) verwiesen. Obwohl XQuery mit dieser Spezifikation eine Turing-vollständige [Kep04] Programmiersprache ist, fehlt noch der wichtige Bereich der Datenmanipulation für den praktischen Einsatz. Die Standardisierung dieses Bereichs ist noch nicht so weit fortgeschritten, so dass sich hier mehrere, zum Teil implementationsspezifische Lösungen gebildet haben. Der vom W3C im Jahr 2011 in der Version 1.0 verabschiedete Standard heißt *XQuery Update Facility* [RCD+11]. Sie wird von dem dem Anwendungsbeispiel zu Grunde liegenden DBMS *eXist-db* derzeit nicht unterstützt.

Aufgrund der lange Zeit nicht vorliegenden Spezifikation verwendet dieses DBMS eine *XQuery Update Extension* [17] genannte Erweiterung. Diese Erweiterung ist konzeptionell der W3C-Spezifikation sehr ähnlich, da die Literatur diese aber nicht umfassend behandelt, wird im Folgenden näher darauf eingegangen. Ausdrücke der XQuery Update Extension haben einen leeren Rückgabe-

```
1  xquery version ”3.0”;
2  module namespace lib=”http://example.org/lib”;
3  declare namespace odm=”http://www.cdisc.org/ns/odm/v1.3”;
4  declare function lib:role-nodes($doc as node()) as
       element() {

5    <RoleData xmlns=”http://www.cdisc.org/ns/odm/v1.3”>
6    {
7    let $users := $doc/odm:AdminData/odm:User
8    for $role in distinct-values($users/@UserType/string())
9    order by $role
10   return
11     <Role Type=”{$role}”>
12     {
13     for $user in $users
14     where  $user/@UserType/string() = $role
15     return
16       <User>
17       {
18       ($user/@OID, $user/*)
19       }
20       </User>
21     }
22     </Role>
23   }
24   </RoleData>
25 };
```

Programm 3.11: Ein Software-Bibliotheksmodul, das die Konvertierung von ODM User-Elementen, wie in Programm 3.3 gezeigt, bereitstellt. Das Ergebnis der Konvertierung ist ein nach Rollentyp sortiertes Format.

```
1 xquery version "3.0";
2 import module namespace lib="http://example.org/lib" at "
      lib.xql";

3 let $doc :=doc("admindata.xml")
4 return
5   lib:role-nodes($doc)
```

Programm 3.12: Ein Hauptmodul, das die Konvertierung des geladenen Dokuments durch die Funktion `lib:role-nodes` des importierten Moduls aufruft.

Tabelle 3.2: Syntax der XQuery Update Extension. Die Begriffe ExprSingle und Expr werden in der Bedeutung von XPath benutzt.

Aktion	Argument
`update insert`	Expr (`into` | `following` | `preceding`) ExprSingle
`update replace`	Expr `with` ExprSingle
`update value`	Expr `with` ExprSingle
`update delete`	Expr
`update rename`	Expr `with` ExprSingle

wert, weisen aber Seiteneffekte in Form der Datenmanipulation auf. Dabei können nur persistente Dokumente manipuliert werden, nicht dynamisch im Rahmen eines Ausdrucks erzeugte. Die Manipulationsausdrücke beginnen mit dem Schlüsselwort `update`, worauf die gewünschte Aktion folgt. Die Syntax dieser ist in Tabelle 3.2 angegeben. Mittels `insert` wird ein Knoten an der bezeichneten Stelle eingefügt, wobei das folgende Schlüsselwort `into`, `preceding` oder `following` angibt, ob die Einfügungen als letzter Kindknoten der durch Expr bezeichneten Stelle oder vor oder nach diesem Knoten vorgenommen wird. Die Aktion `replace` ersetzt die bezeichneten Knoten, `value` ändert ihren Inhalt und `delete` löscht sie. Durch `rename` kann der Name von Elementen oder Attributen geändert werden.

In Programm 3.13 werden Anwendungsbeispiele für alle Aktionen gezeigt. Der Ausdruck in Zeile 1 fügt ein Element `User` ein. Die Einfügeposition wird in Zeile 7 mittels `into` als letztes Kindelement des Knotens `AdminData` bestimmt.

```
1  update insert
2    <User OID="mueller" UserType="Other" xmlns="http://www.
         cdisc.org/ns/odm/v1.3">
3      <FirstName>Eva</FirstName>
4      <LastName>Mueller</LastName>
5      <Email>eva.mueller@example.org</Email>
6    </User>
7  into doc("admindata.xml")/odm:AdminData
8  ------
9  update value doc("admindata.xml")/odm:AdminData/odm:User[
       @OID="mueller"]/odm:Email with "eva@example.org"
10 ------
11 update replace doc("admindata.xml")/odm:AdminData/
       odm:User[@OID = "mueller"]/odm:Email with <odm:Email>
       eva@example.org</odm:Email>
12 ------
13 update delete doc("admindata.xml")/odm:AdminData/odm:User
       [@OID="mueller"]
14 ------
15 update rename doc("admindata.xml)"/odm:AdminData/odm:User
       /odm:FirstName as "odm:LoginName"
```

Programm 3.13: Ausdrücke der XQuery Update Extension, die das bekannte Dokument `admindata.xml` ändern.

In Zeile 9 wird der Wert des Elements `Email` beim Benutzer mit dem Schlüssel `mueller` aktualisiert. Der Ausdruck in Zeile 11 bewirkt in diesem Fall ein identisches Resultat, allerdings wird durch `replace` das gesamte Element `Email` ersetzt. Der in Zeile 13 gezeigte Ausdruck löscht das angegebene Element mittels des Schlüsselworts `delete`. Die Elemente `FirstName`, die in Zeile 15 in der Anweisung `rename` ausgewählt werden, werden durch den Ausdruck in `LoginName` umbenannt. Da hier im Gegensatz zu den vorherigen Ausdrücken keine Einschränkung auf einen bestimmten Wert des Attributs `OID` mittels Prädikat erfolgt, werden sämtliche Elemente `FirstName` durch den Ausdruck erfasst und umbenannt.

Schließlich wird noch ein Überblick über die Fehlerbehandlung mittels Try-Catch-Anweisungen, die seit der Version 3.0 in XQuery spezifiziert sind, gegeben. Führt die Evaluierung eines Ausdrucks zu einem Fehler, so produziert der XQuery-Prozessor einen Fehlerwert mittels der Funktion `error()`. Diese hat

den besonderen Rückgabewert `none`, der nur benutzt wird, um anzuzeigen, dass bei der Evaluierung des Ausdrucks ein Fehler aufgetreten ist und keine Rückgabe geliefert werden kann. Die Funktion `error` bewirkt ebenfalls, dass dem XQuery-Prozessor die Fehlerart mitgeteilt wird. Dazu sind in der XQuery-Spezifikation Fehlercodes beschrieben, zudem können auch programmspezifische Fehlermeldungen ausgegeben werden. Bspw. wird bei der Division durch null der Fehlercode `err:FOAR0001` ausgegeben. Die Eigenschaften eigener Fehlercodes werden der Funktion `error` als Parameter übergeben.

Das Beispiel Programm 3.14 zeigt zur Verdeutlichung der Behandlung unterschiedlicher Fehler eine Funktion, die die durchschnittliche Anzahl der Patienten pro Arzt berechnet und die Anzahl der Patienten und Ärzte als Parameter `$patients` und `$physicians` erwartet. Da die Berechnung des Ausdrucks bei fehlerhaften Eingaben nicht das Anhalten der gesamten Programmausführung bewirken soll, müssen die Programmausführung überwacht und Fehler abgefangen werden. Dies geschieht durch die Verwendung des `try`-Blocks in Zeile 2 mit dem in Zeile 12 folgenden `catch`-Block. Negative Eingabewerte sind aufgrund der Bedeutung auszuschließen, daher wird zunächst geprüft, ob die Parameter negativ sind. Ist das der Fall, wird den Zeilen 5 bis 8 eine eigene Fehlermeldung konstruiert. Als Parameter werden ein qualifizierter Name, eine Fehlerbeschreibung und als Fehlerobjekt die Parameter mit fehlerhaften Werten übergeben. Falls die Eingabeparameter nicht negativ sind, wird das Resultat in Zeile 10 durch Division berechnet und im Element `result` zurückgegeben. Logisch ist es zwar möglich, dass die Anzahl der Ärzte null ist, allerdings kann in diesem Fall kein Verhältnis bestimmt werden. Der Ausdruck zur Berechnung produziert dann aufgrund der Division durch null von selbst einen Fehler. Alle innerhalb des `try`-Blocks entstandenen Fehler werden im nachfolgenden `catch *`-Block behandelt. Die Eigenschaften des Fehlers können über implizit bereitgestellte Variablen des Namensraums `err` abgefragt werden. Daraus wird das Element `error` als Rückgabe konstruiert, das in den Attributen `code`, `description` und `value` den Fehlercode, die textuelle Beschreibung und optional anwendungsspezifische Werte, in diesem Fall die fehlerhaften Parameter des Funktionsaufrufs, enthält.

Aufrufe der Funktion `patient-physician-ratio` und deren Rückgabewerte finden sich in Programm 3.15. Der erste Aufruf mit den Parametern `(200, 10)` liefert das erwünschte Resultat. Der folgende Aufruf weist einen negativen Parameter `(200, -4)` auf, weswegen der anwendungsspezifische Fehler erzeugt wird und ein entsprechendes Fehlerelement zurückgegeben wird. Der letzte Aufruf mit den Parametern `(200, 0)` führt zu einer Division durch null,

```
1  declare function local:patient-physician-ratio($patients
       as xs:int, $physicians as xs:int) as node() {
2  try {
3       if ($patients < 0 or $physicians < 0 )
4       then
5         error(fn:QName("local", "err:NO_NEG"), "Negative
              arguments are not allowed.", (
6           if ($patients < 0 ) then "Patients: " ||
                $patients else (),
7           if ($physicians < 0) then "Physicians: " ||
                $physicians else() )
8         )
9       else
10      <result>{$patients div $physicians}</result>
11 }
12 catch * {
13 <error code="{$err:code}"
       description="{$err:description}"
       value="{$err:value}"/>
14      }
15 };
```

Programm 3.14: Fehlerbehandlung in XQuery mittels Try-Catch-Blöcken.

weswegen der XQuery-Prozessor automatisch einen Fehler produziert, aus dem dann das zurückgegebene Fehlerelement konstruiert wird. Da dieser Fehler der Variable `$err:value` keinen Wert zuweist, bleibt das Attribut `value` darin leer.

Die gezeigten XQuery-Funktionen sind für die Entwicklung von Anwendungslogik grundlegend. Weitere, teilweise implementierungsspezifische Eigenschaften werden bei Bedarf an den entsprechenden Stellen erläutert.

3.3 XForms

Im Folgenden werden die grundlegenden Eigenschaften von XForms als W3C-Standard für die Beschreibung von Formularen vorgestellt.

```
1  ------
2  local:patient-physician-ratio(200, 10)
3  <result>20</result>
4  ------
5  local:patient-physician-ratio(200, -4);
6  <error code="err:NO_NEG" description="Negative arguments
       are not allowed." value="Physicians: -4"/>
7  ------
8  local:patient-physician-ratio(200, 0);
9  <error code="err:FOAR0001" description="Division by zero.
       " value=""/>
```

Programm 3.15: Aufrufe der in Programm 3.14 deklarierten Funktion und die jeweiligen Rückgabewerte.

3.3.1 Konzept

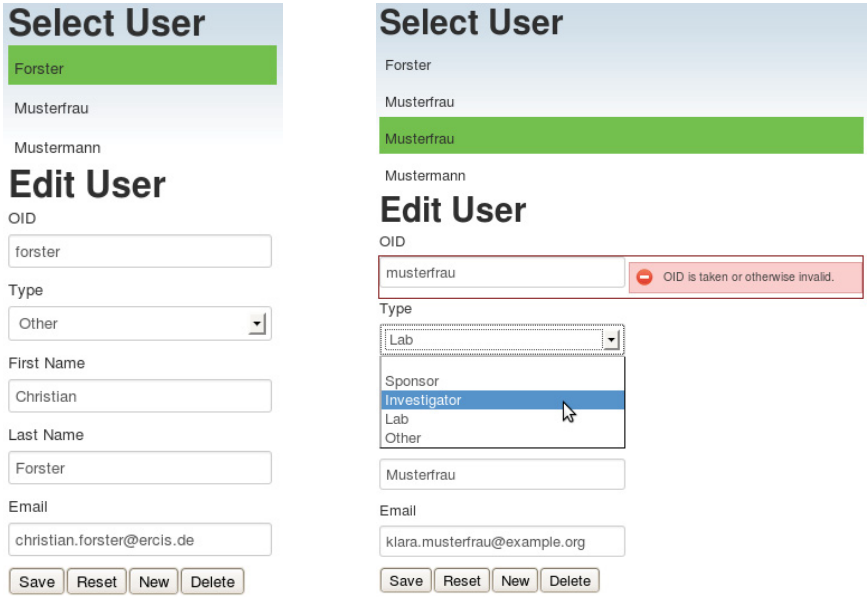
Die derzeit aktuelle Version 1.1 der Spezifikation von XForms [Boy09], auf der diese Ausführungen basieren, stammt aus dem Jahr 2009. XForms basiert auf der Definition von Formularen in einer XML-Sprache, deren Namensraum `http://www.w3.org/2002/xforms` im Folgenden mit dem Präfix `xf` bezeichnet wird. Das damit beschriebene Formular wird durch den jeweiligen XForms-Interpreter dargestellt. Da XForms immer nur die formularspezifischen Aspekte darstellt, wird stets eine umgebende Sprache (host language) benötigt. Ein Konzeptionsmerkmal der Spezifikation ist die Unabhängigkeit von Ausgabegeräten, wodurch es dem XForms-Interpreter überlassen ist, das Aussehen der Elemente zu bestimmen. XForms-Formulare können also bspw. durch einen kompatiblen Browser direkt dargestellt werden, auf Webseiten in HTML-Elemente transformiert werden, auf mobilen Geräten als Elemente der jeweilige Entwicklungsumgebung dargestellt oder sogar durch ein sprachbasiertes System wiedergegeben werden [HP06]. Die Anwendung von XForms im Rahmen dieser Arbeit bezieht sich auf den Einsatz in Webanwendungen, wodurch XForms-Formulare in HTML als host language eingebettet werden. Die Interpretation von XForms geschieht mittels eines serverseitigen Frameworks, das eine automatisierte Transformation der XForms-Formulare in HTML vornimmt, so dass sie durch den Webbrowser dargestellt werden können. Da diese Transformation vollständig durch das Framework übernommen wird, kann bei der Entwicklung vom konkreten XForms-Interpreter abstrahiert werden, solange ausschließlich standardisierte Elemente verwendet werden. Falls in der vorliegenden Arbeit

herstellerspezifische Elemente verwendet werden müssen, weil Funktionen benötigt werden, die nicht standardisiert sind, wird darauf gesondert hingewiesen.

Der konzeptionelle Unterschied zu den in HTML verfügbaren Formularen liegt in der Aufteilung in Datenmodell, Verhalten und Darstellung. XForms verfügt im Gegensatz zu HTML über ein typisiertes Datenmodell, das die Datentypen des XML Schema verwendet und zusätzlich die Definition weiterer Datentypen zulässt. Für die interne Datenrepräsentation und die Kommunikation eines XForms-Formulars mit der Geschäftslogik werden XML-Dokumente anstelle von bei HTML-Formularen genutzten Schlüssel/Wert-Paaren benutzt. Schließlich unterstützt die XForms-Spezifikation den Standard XML Events [MPR03], eine dem Entwurfsmuster Observer [GHJV94] folgende Implementierung zur Behandlung von Ereignissen, mittels dessen das Formularverhalten weitgehend deklarativ konfiguriert werden kann, im Gegensatz zur imperativen Programmierung mittels JavaScript bei HTML-Formularen. Dieses Benachrichtigungsmodell wird auch verwendet, um die Elemente der Darstellung über Änderungen am Datenmodell zu benachrichtigen. Durch diese an das Konzept „Model View Controller (MVC)“ [KP88] angelehnte Struktur ist XForms besonders für die Darstellung komplexer Formulare geeignet.

Anhand eines Formulars zur Bearbeitung von ODM `User`-Elementen werden im Folgenden der Aufbau und die Funktionsweise eines XForms-Formulars erläutert. Es bietet eine Liste aller Benutzer, aus der jeweils einer zur Bearbeitung der Daten im darunterliegenden Formular dargestellt werden kann. Der im Anwendungsbeispiel verwendete XForms-Interpreter betterFORM[7] stellt das Beispiel wie in Abbildung 3.3 gezeigt dar. In Programm 3.16 wird der zugehörige XForms-Programmcode gezeigt, der einen Ausschnitt aus dem umgebenden HMTL-Dokument darstellt. Die folgenden Abschnitte nehmen darauf Bezug.

[7] http://www.betterform.de/



(a) Gültiger Zustand.

(b) Ungültiger Zustand mit Warnhinweis an den Benutzer und mit ausgeklappter Dropdown-Liste.

Abbildung 3.3: Eingabeformular für ODM User-Elemente, dargestellt durch den XForms-Interpreter betterFORM. Der im oberen Bereich grün hinterlegte Listeneintrag ist der für die Bearbeitung ausgewählte Nutzer.

```
1  <xf:model id="m_1" schema="ODM1-3-1-foundation.xsd">
2    <xf:instance src="admindata.xml" id="i_data"/>
3    <xf:instance src="ODM1-3-1-foundation.xsd" id="i_schema
         "/>
4    <xf:instance id="i_template">
5      <User OID="temp" UserType="" xmlns="http://www.cdisc.
           org/ns/odm/v1.3">
6        <FirstName></FirstName>
7        <LastName>New User</LastName>
8        <Email></Email>
9      </User>
10   </xf:instance>
```

```
11    <xf:bind nodeset="instance('i_data')/odm:User/@OID"
          type="odm:oid" constraint="deep-equal(instance('
          i_data')/odm:User/@OID/string(), distinct-values(
          instance('i_data')/odm:User/@OID/string()))"  id="
          b_userOID"/>
12    <xf:bind nodeset="instance('i_schema')/xs:simpleType[
          @name='UserType']/xs:restriction/xs:enumeration" id
          ="b_userType"/>
13    <xf:bind nodeset="instance('i_data')/odm:User[index('
          r_user')]" id="b_activeUser"/>
14    <xf:submission ref="instance('i_data')" resource="
          admindata.xml" method="put" replace="none" id="
          s_save"/>
15    <xf:reset ev:observer="buttonReset" ev:event="
          DOMActivate" model="m_1"/>
16    <xf:insert ev:event="DOMActivate" ev:observer="
          buttonInsert" origin="instance('i_template')" bind=
          "b_activeUser" position="after" if="index('r_user')
           > 0"/>
17    <xf:insert ev:event="DOMActivate" ev:observer="
          buttonInsert" origin="instance('i_template')"
          context="instance('i_data')" if="index('r_user') =
          0"/>
18    <xf:delete ev:observer="buttonDelete" ev:event="
          DOMActivate" bind="b_activeUser"/>
19  </xf:model>
20  <h1>Select User</h1>
21  <xf:repeat id="r_user" nodeset="instance('i_data')//
        odm:User">
22    <xf:output ref="odm:LastName"/>
23  </xf:repeat>
24  <h1>Edit User</h1>
25  <xf:group bind="b_activeUser">
26    <xf:input ref="@OID">
27      <xf:label>OID</xf:label>
28      <xf:alert ev:event="xforms-invalid">OID is taken or
            otherwise invalid.</xf:alert>
29    </xf:input><br/>
30    <xf:select1 ref="@UserType">
31      <xf:label>Type</xf:label>
32      <xf:itemset bind="b_userType">
33        <xf:label ref="@value"/>
```

```
34        <xf:value ref="@value"/>
35      </xf:itemset>
36    </xf:select1><br/>
37    <xf:input ref="odm:FirstName">
38      <xf:label>First Name</xf:label>
39    </xf:input><br/>
40    <xf:input ref="odm:LastName">
41      <xf:label>Last Name</xf:label>
42    </xf:input><br/>
43    <xf:input ref="odm:Email">
44      <xf:label>Email</xf:label>
45    </xf:input>
46  </xf:group>
47  <xf:submit submission="s_save">
48    <xf:label>Save</xf:label>
49  </xf:submit>
50  <xf:trigger id="buttonReset">
51    <xf:label>Reset</xf:label>
52  </xf:trigger>
53  <xf:trigger id="buttonInsert">
54    <xf:label>New</xf:label>
55  </xf:trigger>
56  <xf:trigger id="buttonDelete">
57    <xf:label>Delete</xf:label>
58  </xf:trigger>
```

Programm 3.16: Formularbeschreibung in XForms. Das Formular ermöglicht das Editieren der Eigenschaften eines Users.

3.3.2 Datenmodell

Datenmodell und Verhaltensanweisungen eines XForms-Formulars werden innerhalb des Elements `model` in die umgebende Sprache eingebettet. Das Datenmodell der Beispielanwendung Programm 3.16 befindet sich in den Zeilen 2 bis 13. Das oder die zur Verwendung im XForms-Formular vorgesehenen XML-Dokumente werden entweder explizit als Inhalt des Elements `instance` angegeben oder über dessen Attribute `src` oder `resource` verlinkt. Im Beispiel wird das externe Dokument `admindata.xml` mit bekanntem Inhalt eingebunden. Des Weiteren wird das XSD-Schema `ODM1-3-1-foundation.xsd`, das die von ODM definierten Datentypen enthält, als Instanz eingebunden. Es wird benötigt, um die zulässigen Werte für das Attribut `UserType` auszulesen und per

Auswahlliste anzubieten. Die dritte eingebundene Instanz enthält ein explizit angegebenes Dokument. Es ist die Datenstruktur für einen Benutzer und wird als Vorlage für neu anzulegende Nutzer verwendet. Die Instanzelemente verfügen über das Attribut `id`, das einen Schlüsselwert enthält. Dieses Attribut kann generell in allen zu XForms gehörenden Elementen zur Vergabe eines Schlüssels verwendet werden.

Mittels des Bindungsmechanismus können Eigenschaften an die im Attribut `nodeset` mit XPath gewählte Knoten der XML-Dokumente gebunden werden. Durch die Vergabe eines Schlüssels im Attribut `id` wird außerdem der Zugriff darauf erleichtert. XForms stellt einige Funktionen bereit, mit denen XForms-spezifische Eigenschaften gehandhabt werden, so z. B. die Funktion `instance()`, die verwendet werden kann, um auf das Dokument in einem durch `id` identifizierten Element `instance` zuzugreifen. Einige der vielfachen Einsatzmöglichkeiten von Bindungen werden folgend gezeigt.

So werden im Beispiel in Zeile 11 die Schlüsselattribute `OID` mit weiteren Eigenschaften versehen. Zunächst wird der aus dem ODM XSD-Schema stammende Datentyp `odm:oid` durch das Attribut `type` für gültige Werte zugrunde gelegt. Sodann wird als Wert des Attributes `constraint` ein XPath-Ausdruck als weitere Bedingung für die Gültigkeit angegeben. Es wird sichergestellt, dass kein Schlüsselwert mehrfach auftritt, indem die Liste aller als Schlüssel vorhandenen Werte mit einer Liste verglichen wird, die durch die Funktion `distinct-value` zurückgegeben wird und die daher nur unterschiedliche Werte enthält.

Mittels der Bindung in Zeile 12 wird eine Bindung erzeugt, die einen einfachen Zugriff auf die für das Attribut `UserType` zulässigen Werte ermöglicht, indem sie den Knoten aus den Schemainformationen, der die Auflistung enthält, mit dem Schlüssel `b_userOID` verknüpft.

Die Bindung in Zeile 13 mit der Identität `b_activeUser` stellt die Grundlage für die Auswahl des zu bearbeitenden Nutzereintrags dar. Mittels der Funktion `index('r_user')` wird die aktuelle Position einer in den Darstellungskomponenten zu definierenden Liste abgefragt. Dasjenige Element `User`, das jeweils an dieser Position steht, wird dadurch über diese Bindung auswählbar.

3.3.3 Verhalten

Neben den Instanzen und damit verbundenen Eigenschaften werden im Element `model` noch durch Benachrichtigungen auslösbare Verhaltensweisen deklariert. Im Beispiel geschieht dies durch die Elemente in den Zeilen 14 und 18.

Um Daten zwischen der Präsentationsschicht der Webanwendung und der Geschäftslogik zu übertragen, wird das Element `submission` verwendet. Da-

mit werden Übertragungsarten mit dem Element `submission` festgelegt, die durch Ereignisse ausgelöst werden können. In Zeile 14 findet sich die Übertragung zum Speichern, das zu übertragende Dokument wird durch das Attribut `ref` angegeben. Es können sowohl vollständige Instanzdokumente als auch Teilbäume davon für die Übertragung verwendet werden. Die dabei zu verwendende Methode ist hier HTTP `PUT` und wird im Attribut `method` eingestellt, wobei neben den üblichen REST Methoden weitere implementierungsspezifische existieren können. Es wird angenommen, dass die im Attribut `resource` als Ziel mit der URL `admindata.xml` angegebene Geschäftslogik bei der Übertragungsart PUT das übertragene XML-Dokument entgegennimmt und für eine Persistierung sorgt. Das Attribut `replace` gibt an, ob mit der Antwort des Webservers ein Instanzdokument ersetzt werden soll. Im hier gezeigten Fall wird, da bei der Methode PUT keine Antwort zu erwarten ist, keine Ersetzung vorgenommen.

Das Element `reset`, hier in Zeile 15, dient zum Zurücksetzen des Datenmodells auf den initialen Zustand. Es wird verwendet, um Nutzereingaben zu verwerfen. Durch die Verwendung der Attribute `ev:observer` und `ev:event` aus dem XML Event Namensraum wird festgelegt, wann die Aktion ausgeführt werden soll. Hier geschieht dies, wenn an dem Element, das durch `buttonReset` identifiziert wird, die Aktion `DOMActivate` auftritt. Die Aktion wird also durch einen Klick des Benutzers auf die in der Darstellung anzulegende Schaltfläche ausgelöst.

Die Aktivierung der weiteren Aktionen zum Einfügen und Löschen von Datensätzen wird analog dazu ausgelöst. Das Element `insert` in Zeile 16 fügt Daten in eine bestehende Instanz ein. Dazu fertigt die Aktion eine Kopie der bezeichneten Datenstruktur an und fügt sie in den angegebenen Kontext ein. Das Attribut `origin` gibt dabei die Quelle der Daten an, hier die Instanz mit der Vorlage für die Datenstruktur eines Elements `User`. Das Ziel wird mittels des Attributs `bind` angegeben, in diesem Fall verweist es mit `b_activeUser` auf die Bindung des aktuell ausgewählten Elements. Das Attribut `position` mit dem Wert `after` gibt schließlich an, dass die Einfügung hinter dem Zielknoten erfolgen soll. Dies funktioniert nur, solange noch ein ausgewähltes Element `User` in der Liste vorhanden ist. Um die Ausführung einer Aktion an Bedingungen zu knüpfen, stellt die XForms-Spezifikation das Attribut `if` zur Verfügung. Wenn es angegeben wird, muss der darin befindliche XPath-Ausdruck den booleschen Wert *wahr* aufweisen oder in diesen umwandelbar sein, damit die Aktion ausgeführt wird. Im Beispiel wird dies für eine Fallunterscheidung mit dem Wert von `index('r_user'` benutzt, der größer null ist, falls ein Element der zugrunde liegenden Liste ausgewählt ist, und sonst null. Falls ein Wert der Liste ausge-

wählt ist, wird zum Einfügen die gerade beschriebene Aktion ausgeführt. Falls kein Wert der Liste ausgewählt ist, z. B. weil sie leer ist, wird zum Einfügen das Element `insert` in Zeile 17 verwendet. Statt der Bindung an das ausgewählte Benutzerelement benutzt es das Instanzdokument als Kontext und fügt direkt dort ein.

Das Element `delete` in Zeile 18 bewirkt das Löschen des gebundenen Elements. Die Bindung funktioniert analog. Allerdings muss hier keine Fallunterscheidung durchgeführt werden, da im Fall eines nicht ausgewählten Elements auch keine Löschaktion durchgeführt werden kann.

3.3.4 Darstellung

Die im Vorherigen gezeigten Elemente legen das Datenmodell und seine Eigenschaften fest und steuern das Verhalten des Formulars. Es spielt für die Funktionalität keine Rolle, an welcher Stelle der umgebenden Sprache sie stehen, da sie keine Auswirkungen auf die Darstellung des Formulars nehmen. Die Elemente der Darstellung werden hingegen an ihrer Position durch den XForms-Interpreter mittels passender Elemente der umgebenden Sprache abgebildet.

Das Beispiel besteht aus zwei Gruppen von Benutzerelementen, nämlich der Auswahlliste und dem Bearbeitungsformular, die durch die HTML-Überschriftenelemente `h1` in den Zeilen 20 und 24 eingeleitet werden.

Die Auswahlliste wird in Zeile 21 durch das Element `repeat` konstruiert. Es führt eine Iteration über die im Attribut `nodeset` durch den XPath-Ausdruck angegebenen Knoten durch, hier also über jedes Element `User`. Dieses stellt den Kontext für Auswertungen innerhalb des Elements `repeat` dar. Das enthaltene Element `output` gibt den Inhalt des referenzierten Knotens wieder, hier also den Wert des jeweilig iterierten Elements `LastName`. Es entsteht also eine Liste von Nachnamen. Durch die Verknüpfung mit den Elementen des Datenmodells wird diese Liste bei Änderungen der Daten automatisch angepasst. Falls also ein Element `User` gelöscht wird, verschwindet der Name aus der Liste. Beim Einfügen eines solchen Elementes erscheint er und Änderungen des Werts werden ebenfalls berücksichtigt. Durch den Benutzer kann ein Element der Liste ausgewählt werden. Die Position des selektierten Elements wird durch die Bindung in Zeile 13 ausgewertet.

Sie ist relevant für das Bearbeitungsformular, dessen Eingabefelder durch das Element `group` in Zeile 25 umschlossen werden. Dessen Attribut `bind` gibt den Kontext der Gruppierung an, wobei mit dem Wert `b_activeUser` die Bindung referenziert wird, die das jeweils in der Auswahlliste selektierte Element `User` enthält. Die Attribute `ref` der in der Gruppierung liegenden Eingabeelemente

verweisen daher stets auf Werte des aktuell selektierten Eintrags. Es werden das Element `input` zur Eingabe von textuellen Werten und das Element `select1` zur Auswahl eines aus mehreren vorgegebenen Werten verwendet. Das Element `input` in Zeile 26 dient der Eingabe des Schlüsselwertes und wird durch die im Element `label` angegebene Beschriftung in der Benutzeroberfläche gekennzeichnet. Da der Schlüsselwert im Datenmodell mit einer Bedingung für seine Gültigkeit versehen wurde, kann hier zwischen zulässigen und unzulässigen Werten unterschieden werden. Falls der eingegebene Wert unzulässig ist, wird durch das Datenmodell die Benachrichtigung `xforms-invalid` ausgegeben. Das Element `alert` in Zeile 28 reagiert darauf und erzeugt eine Benachrichtigung an diesem Eingabefeld, die den Benutzer auf einen unzulässigen Wert hinweist. Ein Eingabefeld, das die Auswahl eines Wertes aus einer Liste zulässt, wird für das Angeben des Benutzertyps benötigt. Das in Zeile 30 verwendete Element `select1` leistet dies. Die Auswahlmöglichkeiten werden mittels des Elements `itemset` angegeben. Dieses verweist auf die Bindung `b_userType`, die die laut ODM-Spezifikation zulässigen Werte des Schemas enthält. Diese werden dem Nutzer textuell angezeigt, was durch das Element `label` veranlasst wird, und bei Auswahl als Wert in das Datenmodell übertragen, was durch das Element `value` bewirkt wird. Da XForms keine Vorgaben zur konkreten Umsetzung der Eingabeelemente macht, liegt die Darstellung der Eingabeelemente in der Entscheidung des Interpreter. Der hier verwendete XForms-Interpreter stellt die Auswahlliste als Dropdown-Liste dar, alternativ sind auch Radiobuttons denkbar. Die Funktionsweise der noch folgenden Elemente `input` für die Eingabe von Vor- und Nachnamen sowie E-Mail-Adresse ist offensichtlich. Auf die Eingabeelemente folgen ab Zeile 47 Schaltflächen, mit denen verschiedene Aktionen ausgelöst werden können. Durch das Element `submit` lassen sich im Element `model` deklarierte Übertragungen auslösen, hier `s_save`, die das Datenmodell an die Geschäftslogik überträgt. Im Gegensatz zu `submit` stellt das Element `trigger` eine Schaltfläche bereit, die für unterschiedliche Zwecke genutzt werden kann. Ein Klick darauf löst die Benachrichtigung `DOMActivate` aus, die von den Beobachtern der Elemente registriert wird und das im vorherigen Abschnitt beschriebene Verhalten bewirkt.

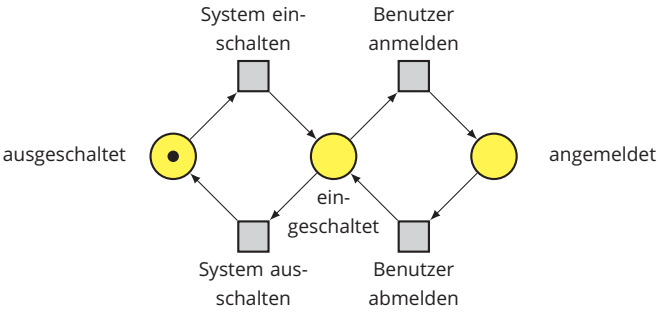


Abbildung 3.4: Petri-Netz, das die Zustandsübergänge eines einfachen Systems darstellt. Man kann es einschalten und dann einen Benutzer an- und wieder abmelden. Ausschalten kann man es nur, wenn der Benutzer nicht angemeldet ist.

3.4 Weitere verwendete Techniken

3.4.1 XML-Netze

Zur Modellierung von Prozessen zu wissenschaftlichen und kommerziellen Zwecken sind Petri-Netze [Pet62] ein etabliertes Werkzeug [SVOK11]. Petri-Netze bestehen aus Stellen und Transitionen, die durch Kreise und Rechtecke dargestellt werden. Sie sind in alternierender Reihenfolge durch gerichtete Kanten verbunden, so dass das Petri-Netz ein bipartiter Graph ist. Transitionen bewirken Übergänge des Netzes zwischen Zuständen entlang der Kanten. Stellen repräsentieren Zustände, sie können durch das Zeichen • gekennzeichnete Marken aufnehmen. Das Aktivieren einer Transition bewirkt, dass eine Marke aus der oder den Stellen im Vorbereich der Transition entfernt wird und eine Marke in die Stelle oder Stellen im Nachbereich hinzugefügt wird. Transitionen lassen sich also nur aktivieren, wenn alle Stellen im Vorbereich eine Marke enthalten. Abbildung 3.4 zeigt die Elemente anhand eines einfachen Petri-Netzes.

Auf Basis der oben beschriebenen Stellen/Transitions-Netze sind Erweiterungen entstanden. Bspw. gefärbte Petri-Netze [Jen87], bei denen die Marken unterscheidbar sind oder die hier verwendeten XML-Netze [LO01; LO03], bei denen die Stellen die Bedeutung eines Dokumentenspeichers haben und Objekte, die durch XML-Dokumente beschrieben werden, aufnehmen. Den Stellen werden XML Schemata zugewiesen, womit die Art der Dokumente spezifiziert ist.

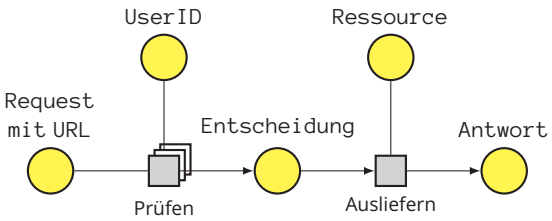


Abbildung 3.5: XML-Netz, das den Vorgang der Rechtekontrolle bei Zugriff auf eine Ressource zeigt. Durch die Schattierung der Aktivität *Prüfen* wird signalisiert, dass dafür eine verfeinerte Darstellung verfügbar ist.

Transitionen werden als Aktivitäten, die XML-Dokumente konsumieren und produzieren, verstanden. Zusätzlich zu gerichteten Kanten werden ungerichtete Kanten erlaubt. Sie bedeuten, dass eine Aktivität nur lesend auf ein Dokument zugreift. Um auf unterschiedlichen Abstraktionsstufen zu modellieren, lassen sich Aktivitäten verfeinern. Im abstrakteren Modell wird eine Aktivität dazu durch ein doppelt schattiertes Aktivitätssymbol gekennzeichnet. Das verfeinerte Modell stellt diese Aktivität detailliert wiederum als XML-Netz dar. Die Objektspeicher im Vor- und Nachbereich der Aktivität müssen in der verfeinerten Darstellung identisch zu denen der höheren Abstraktion sein, sie werden durch ein umschließendes Quadrat deutlich gemacht. Um eine übersichtlichere Darstellung mit weniger Kantenüberschneidungen zu erhalten, können identische Objektspeicher mehrfach abgebildet werden, dazu wird der Umriss gestrichelt. Abbildung 3.5 zeigt ein XML-Netz, das eine Rechteprüfung modelliert. Die Aktivität *Prüfen* erhält in einem Request die URL einer Ressource, auf die der Benutzer, der in den Sitzungsdaten hinterlegt ist, zugreifen möchte. Abhängig vom Prüfergebnis wird die Ressource oder eine Erklärung, warum der Zugriff verweigert wurde, durch die Aktivität *Ausliefern* zurückgegeben.

Abbildung 3.6 zeigt die Verfeinerung der Aktivität *Prüfen*. Sie funktioniert nach dem Prinzip der attributbasierten Zugriffskontrolle [PDMP05]. Für die Prüfung, ob ein Nutzer auf eine URL zugreifen darf, werden die für den Zugriff vorgesehenen Regeln mit den Attributen des Benutzers angewendet. Dazu werden zunächst die einzelnen Zugriffsregeln identifiziert, die angeben, welche Attribute für die Prüfung benötigt werden und welche Attributwerte vorhanden sein müssen. Aus dem gesamten Regelsatz wird dabei eine Referenz auf die zu berücksichtigenden Regeln erzeugt. Über die in der Sitzung gespeicherte Nutzeridentifikation wird die zum Benutzer gehörende Entität, die dessen Attribute

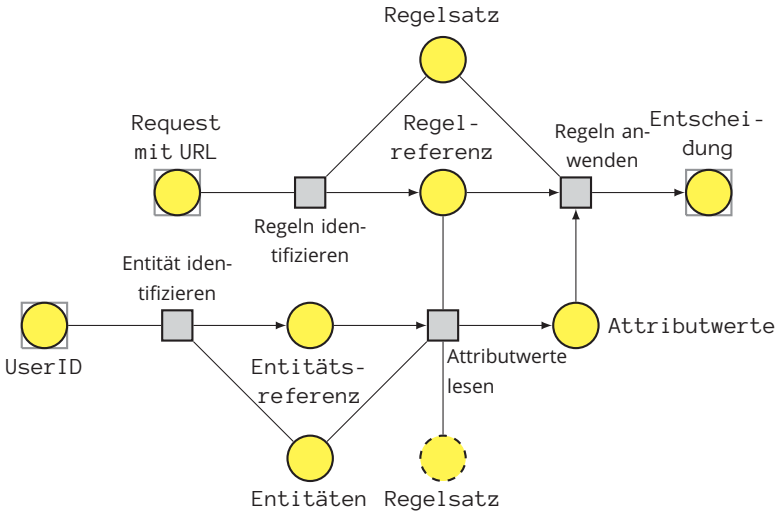


Abbildung 3.6: Verfeinerung der Aktivität *Prüfen*. Die mit einem Quadrat umschlossenen Objektspeicher sind auch im übergeordneten Prozess modelliert und bilden die Anknüpfungspunkte der Verfeinerung.

enthält, identifiziert. Sodann werden die laut Regeln benötigten Attributwerte aus der Entität gelesen. Auf diese werden die Regeln angewendet und es erfolgt eine Entscheidung, ob der Zugriff gestattet oder verweigert ist. Für sämtliche Stellen lassen sich XML Schema-Dateien angeben, damit diese als Objektspeicher interpretierbar sind. Die Verwendung eines Objektspeichers sagt noch nichts über die tatsächliche Persistierung eines Objekts aus. Objektspeicher können sowohl im Sinne persistenter Datenspeicher, wie hier `Regelsatz` oder `Entitäten`, als auch im Sinne temporärer Puffer, wie hier `Entitätsreferenz`, `Regelreferenz` oder `Attributwerte`, benutzt werden.

Weitergehende Spezifizierungen, wie sie typischerweise während der Entwurfsphase benötigt werden, lassen sich durch „Filterschemata“, „Inschriften“ und „Objektmodelle“ [SVOK11] ergänzen. Filterschemata enthalten Verarbeitungsanweisungen, die an den Kanten des XML-Netzes angebracht werden und ausdrücken, wie die Dokumente beim Schalten der Aktivität verarbeitet werden. Mit Inschriften können Bedingungen, von denen die Durchführung der Aktivität abhängig ist, ausgedrückt werden. Objektmodelle werden den Objektspeichern zugewiesen und drücken aus, für welche Art von Dokumenten sie

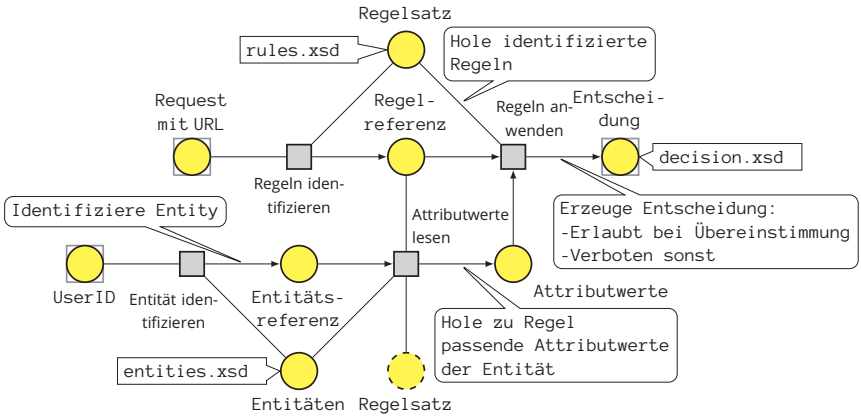


Abbildung 3.7: Weitere Verfeinerung der Aktivität *Prüfen*. Diese Erweiterung von Abbildung 3.6 zeigt Filterschemata an Kanten und XML Schema an Objektspeichern.

vorgesehen sind. Die Abbildung 3.7 erweitert die Darstellung um Filterschemata an Kanten und XML Schema an Objektspeichern. Die Modellierung der Präzisierungen kann mit unterschiedlichen Techniken erfolgen. Eine Vorstellung und Diskussion verschiedener Methoden folgt in Abschnitt 4.3.

3.4.2 XSLT

Mit Extensible Stylesheet Language Transformations (XSLT) steht neben XQuery eine weitere Turing-vollständige [Kep04] Sprache für die Verarbeitung von XML zur Verfügung. Die Unterschiede der beiden Sprachen sind in ihrer Herkunft begründet. Während XQuery aus dem Bereich der Datenbanken stammt und sich an SQL anlehnt, hat XSLT seinen Ursprung in der Verarbeitung von XML-Dokumenten mit dem Zweck sie z. B. als HTML im Web zu publizieren oder in ein sonstiges Format umzuwandeln.

Derzeit liegt XSLT in der Version 2.0 vor [Kay07]. Eine vollständige Beschreibung findet sich in der Literatur [Kay08; Tid08]. Im Folgenden wird das Funktionsprinzip erläutert und durch Beispiele verdeutlicht.

Die Syntax von XSLT-Programmen ist eine XML-Sprache, die von einem XSLT-Prozessor interpretiert wird. XSLT und XQuery basieren beide auf XPath und teilen somit das Datenmodell XDM und die XPath-Funktionen und -Operatoren.

```
1  <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.
       org/1999/XSL/Transform">
2    <xsl:template match="/">
3      <xsl:copy-of select="."/>
4    </xsl:template>
5  </xsl:stylesheet>
```

Programm 3.17: XSLT-Ausdruck, der auf das Dokument in Programm 3.3 (Seite 67) angewendet wird und dessen Inhalt wiedergibt. Sein Wert entspricht dem der XQuery-Implementierung in Programm 3.7.

Kernbestandteil von XSLT sind *Templates* genannte Regeln, die deklarieren, wie ein Element des Eingabedokumentes für die Ausgabe transformiert werden soll. Ein XSLT-Prozessor parst das XML-Eingabedokument und wendet passende Templates auf die vorgefundenen Elemente an. Durch diese Transformation der einzelnen Elemente, die nicht von der Position des Elements im Dokument, sondern vom Zutreffen des Templates abhängt, wird das Ausgabedokument generiert. Dadurch ist ein XSLT-Programm im Vergleich zu XQuery weniger von der Struktur des Eingabedokumentes abhängig, aber Optimierungen durch den XSLT-Prozessor sind kaum möglich [Kay05]. XSLT-Transformationen eignen sich daher konzeptionell eher für Anwendungen, bei denen im Wesentlichen die Elemente des Eingabedokumentes zu einem Ausgabedokument transformiert werden sollen, während XQuery sich eher für Anwendungen eignet, die Daten aus umfangreichen XML-Dokumenten verarbeiten sollen [Kay05]. Aus syntaktischer Sicht ist die XSLT-Implementierung eines Programms aufgrund ihres XML-Formats meist weniger kompakt als die XQuery-Implementierung.

Die im Folgenden vorgestellten XSLT-Transformationen führen zum gleichen Ergebnis wie die in Programm 3.7 (Seite 72) bis 3.9 gezeigten XQuery-Ausdrücke.

Programm 3.17 zeigt die Grundstruktur einer XSLT-Transformation, die mit der Angabe des Elements `stylesheet` beginnt. Es werden die XSLT-Version `2.0` und der Namensraum mit dem Präfix `xsl` angegeben. Es folgt das Element `template`, um das Template anzugeben, das auf das Wurzelelement `/` des Eingabedokumentes angewendet wird. Das Element `copy-of` (Zeile 3) führt eine Kopie des referenzierten Elements durch, die auch Attribute und Kindelemente umfasst. Das Programm gibt also das Eingabedokument unverändert aus.

Programm 3.18 ist ähnlich aufgebaut, aber der Inhalt des Templates in den Zeilen 3 bis 6 ist umfangreicher. Mittels der Anweisung `for-each` wird eine Iteration über die im Attribut `select` angegebenen Elemente `User` durchgeführt.

```
1  <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.
       org/1999/XSL/Transform" xmlns:odm="http://www.cdisc.
       org/ns/odm/v1.3">
2    <xsl:template match="/">
3      <xsl:for-each select="odm:AdminData/odm:User">
4        <xsl:sort select="odm:FirstName"/>
5        <xsl:copy-of select="."/>
6      </xsl:for-each>
7    </xsl:template>
8  </xsl:stylesheet>
```

Programm 3.18: XSLT-Ausdruck, der auf das Dokument in Programm 3.3 (Seite 67) angewendet wird und eine Sequenz der Elemente `User`, sortiert nach Vornamen, zurückgibt. Dies entspricht der in Programm 3.8 in XQuery gezeigten Implementierung.

Die dabei durchlaufenen Elemente werden durch die Anweisung `sort` gemäß ihres Wertes in `FirstName` sortiert und schließlich durch die bereits bekannte Anweisung `copy-of` ausgegeben. Somit entsteht als Ausgabe eine sortierte Sequenz der Elemente `User`.

Programm 3.19 besteht aus zwei Templates, die sich in den Zeilen 2 bis 13 und 14 bis 17 befinden. Das erste Template wird wieder auf das Wurzelelement / angewendet. Darin wird zunächst mit der Anweisung `variable` eine Referenz auf das Element `AdminData` angelegt. In Zeile 4 folgt dann die direkte Angabe des neuen Wurzelelements `RoleData` im Ausgabedokument. Als Nächstes wird eine Iteration über die unterschiedlichen Werte des Attributes `UserType` durchgeführt, um die Gruppierung nach Rolle zu bilden. Die Rollen werden mittels der Anweisung `sort` sortiert (Zeile 6) und die in der Iteration verwendete Rolle wird zum späteren Zugriff in der Variable `role` gespeichert (Zeile 7). Dann wird ein Element `Role`, das den Typ der Rolle im Attribut `Type` enthält, angelegt. In diesem Element (Zeile 9) erfolgt mittels `apply-templates` eine erneute Suche nach einem passenden Template und dessen Anwendung auf die durch das Attribut `select` angegebenen Elemente `User`, die den in dieser Iteration behandelten Typ aufweisen. Die Wahl des Template fällt nun auf das für `User` passende zweite Template in Zeile 14, das das Element `User` im Ausgabedokument konstruiert. Es soll den Inhalt des Elements `User` aus dem Eingabedokument aufweisen, aber nicht über dessen Attribut `UserType` verfügen, da der Typ bereits durch das umgebende Element `RoleData` bestimmt ist. Dazu enthält es

```
1  <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.
       org/1999/XSL/Transform" xmlns:odm="http://www.cdisc.
       org/ns/odm/v1.3" exclude-result-prefixes="odm">
2    <xsl:template match="/">
3      <xsl:variable name="admindata" select="odm:AdminData"
           />
4      <RoleData xmlns="http://www.cdisc.org/ns/odm/v1.3">
5        <xsl:for-each select="distinct-values(odm:AdminData
             /odm:User/@UserType/string())">
6          <xsl:sort select="."/>
7          <xsl:variable name="role" select="."/>
8          <Role Type="{$role}">
9            <xsl:apply-templates select="$admindata/
                 odm:User[@UserType=$role]"/>
10         </Role>
11       </xsl:for-each>
12     </RoleData>
13   </xsl:template>
14   <xsl:template match="odm:User">
15     <xsl:copy>
16       <xsl:attribute name="OID"><xsl:value-of select="
             @OID"/></xsl:attribute>
17       <xsl:copy-of select="*"/>
             @*</xsl:copy></xsl:template>(*@
18 </xsl:stylesheet>
```

Programm 3.19: XSLT-Ausdruck, der auf das Dokument in Programm 3.3 (Seite 67) angewendet wird und die Nutzer nach Rollen gruppiert zurückgibt. Das Ergebnis entspricht dem in Programm 3.9 gezeigten XQuery-Beispiel.

eine Anweisung `copy`, die im Gegensatz zur bekannten Anweisung `copy-of` nur das Element ohne Attribute und Kindelemente kopiert. Mittels der Anweisung `attribute` (Zeile 16) wird ein Attribut `OID` mit dem durch `value-of` bestimmten Wert hinzugefügt und schließlich werden alle Kindelemente mittels `copy-of` (Zeile 17) kopiert. Die Ausführung der Transformation führt also zu dem bereits aus Programm 3.10 (Seite 74) bekannten Dokument.

Die Beispiele illustrieren, wie mit der Verwendung von XSLT Funktionalität realisiert werden kann, die zuvor mittels XQuery implementiert wurde. Durch

die XML-Syntax von XSLT ist der Programmcode tendenziell umfangreicher. Für die hier vorgestellte XML-basierte Architektur wird primär XQuery verwendet, für die generative Erstellung von XForms-Formularen, deren Struktur sich aus den ODM Metadaten ableitet, wird auf XSLT zurückgegriffen.

3.4.3 XSL FO

XML-Dokumente werden zwar aufgrund der Verwendung von Textdateien als Repräsentationsformat häufig als menschenlesbar bezeichnet, allerdings ist damit eher die grundlegende Eigenschaft gemeint, dass ein XML-erfahrener Domänenexperte ein Dokument in dieser Form verstehen kann, und nicht die komfortable Benutzbarkeit eines Dokumentes durch alle Benutzer. Hierzu wird eine Transformation benötigt, mittels derer ein XML-Dokument in die klassische, seitenorientierte Darstellung eines papierbasierten Dokuments umgewandelt werden kann. Die vom W3C spezifizierte Sprache XSL Formatting Objects (XSL FO) [Ber06] beschreibt die Layoutelemente, mit denen die Inhalte eines XML-Dokuments in ein seitenorientiertes Dokument transformiert werden können. Dies umfasst neben offensichtlich benötigten Festlegungen von Schriftart und -größe auch umfangreiche Satzanweisungen für die Seitenaufteilung, die Silbentrennung und die Bestimmung von Kopf- und Fußzeile, die für ein hochwertiges Dokument benötigt werden [Lov02]. Diese FO Templates werden von einem *Formatter* genannten Programm verarbeitet, das das Rendering des Ausgabedokuments übernimmt. Eine freie Implementierung ist bspw. das Programm Apache FOP[8].

Das folgende Beispiel Programm 3.20 illustriert die grundlegende Funktionsweise von XSL FO. Der XQuery-Code erzeugt ein XSL FO-Template und wendet es auf das aus Programm 3.3 bekannte Dokument mit Personendaten an, um eine tabellarische Darstellung zu erzielen. Der Namensraum `fo` enthält die durch XSL FO definierten Elemente.

Der Schwerpunkt liegt auf der Erzeugung des XSL FO-Templates, die in Zeile 2 mit dem Element `layout-master-set` beginnt. Darin wird in Zeile 4 das Seitenlayout des Zieldokuments mit dem Namen `pdf-report` festgelegt. Dabei werden der Hauptbereich der Seite als `region-body` sowie Kopf- und Fußzeile als `region-before` und `region-after` bestimmt. Das in Zeile 12 verwendete Element `page-sequence` stellt einen Container für den Inhalt des Dokuments dar. Es verweist auf die zuvor als `pdf-report` vorgenommenen Seitendefinitionen. Als Seiteninhalt wird in Zeile 13 die Kopfzeile mit dem Titel `Personenre-`

[8] Apache FOP (Formatting Objects Processor)https://xmlgraphics.apache.org/fop/

gister und dem Erstellungsdatum, das dynamisch per XQuery bestimmt wird, eingefügt. In Zeile 19 wird die Fußzeile bestimmt, die die Angabe der Seitenzahl enthält. Die wird aufgrund des Elements page-number vom XSL FO-Formatter dort eingefügt. Schließlich wird in Zeile 24 der Hauptbereich der Seite für den Tabelleninhalt innerhalb eines flow-Elements bestimmt. Während Kopf- und Fußzeile statische Positionen einnehmen, stellt dieses Element einen durch den Formatter zu platzierenden Inhalt dar. So erfolgen in diesem Bereich bspw. automatische Seitenumbrüche. Das Element table in Zeile 24 umfasst die Erzeugung der Tabelle. Zunächst werden die Spalten und ihre Breite definiert, dann wird in Zeile 29 eine Tabellenzeile als Tabellenkopf ausgewiesen. Sie enthält die Spaltentitel und wird auf jeder Seite wiederholt. Ab Zeile 45 wird im Element table-body der Tabelleninhalt erzeugt. Hierzu wird mit XQuery über die einzelnen User-Elemente des ODM-Dokuments iteriert und für jedes Element eine Tabellenzeile erzeugt, in die die Daten zu Vor- und Nachname, E-Mail-Adresse und Rolle eingetragen werden.

Das so erzeugte Template wird in Zeile 68 an den XSL FO-Formatter übergeben, wobei das Zielformat als application/pdf bestimmt wird, um ein PDF-Dokument zu erzeugen. Generell sind je nach Fähigkeiten des Formatters auch weitere Zielformate möglich. Das davon erzeugte Dokument wird in der folgenden Zeile als Download ausgegeben.

```
1  let $doc := doc('admindata.xml')
2  let $fo-template :=
3  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
4    <fo:layout-master-set>
5      <fo:simple-page-master master-name="pdf-report">
6        <fo:region-body margin-top="3.5cm" margin="2.5cm"/>
7        <fo:region-before extent="2.5cm" display-align="
             before"/>
8        <fo:region-after extent="2.5cm" display-align="
             after"/>
9      </fo:simple-page-master>
10   </fo:layout-master-set>
11
12   <fo:page-sequence master-reference="pdf-report">
13     <fo:static-content flow-name="xsl-region-before">
14     <fo:block margin="1cm">
15       <fo:block font-size="18pt">Personenregister</
             fo:block>
16       <fo:block>Erzeugt am: { fn:format-date(fn:current-
             date(), "[D01].[M01].[Y]") }</fo:block>
```

```
17      </fo:block>
18      </fo:static-content>
19      <fo:static-content flow-name="xsl-region-after">
20        <fo:block margin="1cm">Seite <fo:page-number/></
              fo:block>
21      </fo:static-content>
22      <fo:flow flow-name="xsl-region-body">
23        <fo:block>Registrierte Benutzer</fo:block>
24        <fo:table border="solid" border-collapse="collapse"
               margin-top="12pt">
25          <fo:table-column column-width="20%"/>
26          <fo:table-column column-width="20%"/>
27          <fo:table-column column-width="40%"/>
28          <fo:table-column column-width="20%"/>
29          <fo:table-header border-bottom-style="solid">
30            <fo:table-row >
31              <fo:table-cell padding="2pt">
32                <fo:block>Name</fo:block>
33              </fo:table-cell>
34              <fo:table-cell padding="2pt">
35                <fo:block>Vorname</fo:block>
36              </fo:table-cell>
37              <fo:table-cell padding="2pt">
38                <fo:block>E-Mail</fo:block>
39              </fo:table-cell>
40              <fo:table-cell padding="2pt">
41                <fo:block>Rolle</fo:block>
42              </fo:table-cell>
43            </fo:table-row>
44          </fo:table-header>
45          <fo:table-body>
46          { for $contact in $doc/odm:AdminData/odm:User
47          return
48            <fo:table-row border-bottom-style="dashed">
49              <fo:table-cell padding="2pt">
50                <fo:block>{ $contact/odm:LastName/text() }<
                      /fo:block>
51              </fo:table-cell>
52              <fo:table-cell padding="2pt">
53                <fo:block>{ $contact/odm:FirstName/text() }
                      </fo:block>
54              </fo:table-cell>
```

```
55              <fo:table-cell padding="2pt">
56                <fo:block>{ $contact/odm:Email/text() }</
                      fo:block>
57              </fo:table-cell>
58                <fo:table-cell padding="2pt">
59                <fo:block>{ $contact/@UserType/string() }</
                      fo:block>
60              </fo:table-cell>
61            </fo:table-row> }
62          </fo:table-body>
63        </fo:table>
64      </fo:flow>
65    </fo:page-sequence>
66  </fo:root>
67
68  let $pdf := xslfo:render($fo-template, "application/pdf",
        ())
69  return response:stream-binary($pdf, "application/pdf", "
        personenliste.pdf")
```

Programm 3.20: Erzeugung einer XSL FO-Transformation, die eine benutzerfreundliche Tabelle aus ODM-Stammdaten generiert.

Abbildung 3.8 zeigt das erzeugte Dokument. Das Beispiel dient der Illustration der grundlegenden Funktionsweise von XSL FO. Der Standard bietet darüber hinausgehend umfangreiche Möglichkeiten zur Definition von seitenorientierten Dokumenten. Weitreichende Beschreibungen sind bspw. in [Lov02] oder [Paw02] zu finden.

3.4.4 CDISC ODM

Bei der Durchführung klinischer Studien werden in zunehmendem Maß Softwaresysteme zur Dokumentation verwendet. Diese auch Electronic Data Capture (EDC) genannten Systeme ersetzen papierbasierte Dokumentationsformulare und kommen in ca. der Hälfte der durchgeführten Studien zum Einsatz [14]. Sie bilden die im *Studienprotokoll* festgelegten Schritte ab und legen die zu erhebenden Daten fest. Je nach Studie kommen dabei sehr unterschiedliche EDCs zum Einsatz, neben Closed- und Open-Source-Systemen auch Eigenentwicklungen [KOY+10]. Um die Interoperabilität zwischen unterschiedlichen Systemen zu erhöhen und sowohl Studienprotokolle, die die Metadaten einer Studie darstellen, als auch die in einer Studie erhobenen Daten in unterschiedlichen Systemen ver-

Personenregister

Erzeugt am: 11.01.2014

Registrierte Benutzer

Name	Vorname	E-Mail	Rolle
Forster	Christian	christian.forster@ercis.de	Other
Musterfrau	Petra	petra@example.org	Investigator
Mustermann	Max	max@example.org	Investigator

Seite 1

Abbildung 3.8: Durch XSL FO erzeugtes Dokument, das die ODM-Benutzerdaten aus Programm 3.3 als seitenorientiertes, benutzerfreundliches Dokument darstellt.

wenden zu können, wurde der XML-basierte Standard *Operational Data Model (ODM)* [CDI10] von der Organisation *Clinical Data Interchange Standards Consortium (CDISC)*[9] entwickelt. Er liegt derzeit in der im Jahr 2010 verabschiedeten Version 1.3.1 in textueller Beschreibung und als XML Schema vor. Eine wichtige Eigenschaft der ODM-Spezifikationen ist die Erweiterungsfähigkeit. Um Implementierungen, die als *Vendor Extensions* über den allgemein spezifizierten Teil hinaus gehen, zu erleichtern, bindet das Schema für jedes Element eine Erweiterungsgruppe ein, die durch die Implementierung mit Inhaltsdefinitionen angereichert werden kann und weiteren zulässigen Inhalt eines Elementes angibt. Die Erweiterungen müssen in einem jeweils separaten Namensraum angelegt sein, damit Erweiterungen mehrerer Hersteller nicht zu Konflikten führen. Die während des Single-Source-Projekts entwickelte Software x4T basiert auf der genannten Version von ODM und nutzt Vendor Extensions.

ODM – Grundstruktur Bei ODM-Dokumenten existieren zwei Typen: *Snapshot* und *Transactional*. Snapshot-Dokumente geben den derzeit gültigen Stand der Studie wieder, während Transactional-Dokumente den historischen Verlauf wiedergeben. Im Rahmen des Single-Source-Projekts wurde die Snapshot-Variante verwendet, weswegen diese hier vorgestellt wird. Die wesentlichen in ODM vorgesehenen Elemente dienen der Festlegung des Studienprotokolls (Study), der Angabe von administrativen Daten (AdminData) und der Angabe von Patientendaten (ClinicalData). Die Umsetzung des Studienprotokolls in ODM geschieht außerhalb des EDC durch einen speziellen Studieneditor oder einen generischen XML-Editor. Ein kompatibles EDC verwendet diese Datei und nutzt die Studien-Metadaten zur Erstellung der Formulare für jeden Studienteilnehmer. Um einen Überblick über die im Folgenden anhand von XML erläuterte ODM-Struktur zu geben, enthält Abbildung 3.9 eine Visualisierung.

Das Grundgerüst einer ODM-Datei ist in Programm 3.21 gezeigt. Das Wurzelelement `ODM` enthält einige obligatorische Attribute, die die Datei näher bestimmen. Es folgt das Element `Study` (Zeile 7), das das Studienprotokoll enthält. Zunächst erfolgen global gültige Beschreibungen im Element `GlobalVariables` (Zeile 8) und ggf. grundlegende Definitionen von Maßeinheiten unter `BasicDefinitions` (Zeile 13). Hier sind dies Informationen zur Beispielstudie und die Definition der Maßeinheit für Blutdruck, auf die später zurückgegriffen wird. Sodann erfolgt die Definition der Dokumentation und ihrer Komponenten im Element `MetaDataVersion` (Zeile 20). Da der Inhalt dieses Elements umfangreich ist, wird er gesondert in Programm 3.22 dargestellt und erklärt. Mit

[9] http://www.cdisc.org

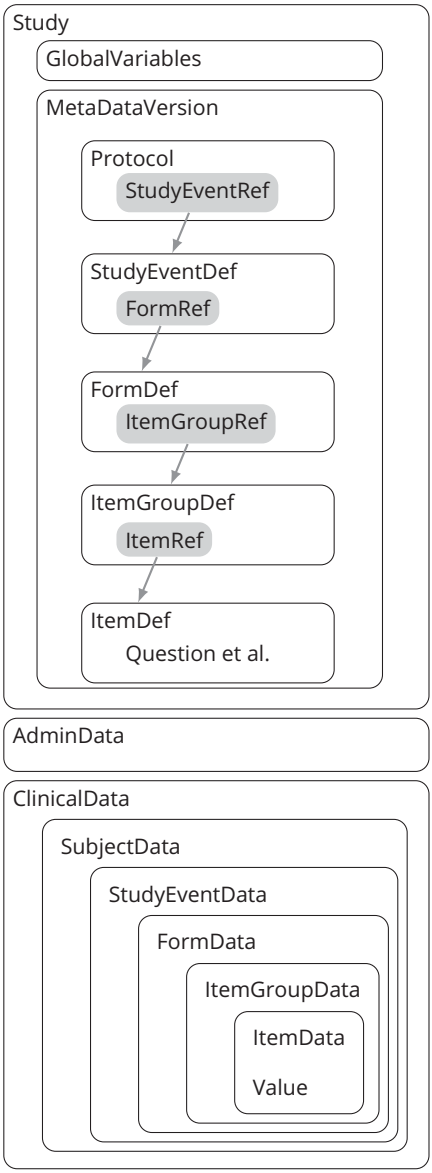


Abbildung 3.9: ODM-Struktur, angelehnt an [15].

Tabelle 3.3: Elemente des Studienprotokolls in ODM.

Element	Bedeutung	Beispiel
`MetaDataVersion`	Versionstand	MD.1.0
`StudyEvent`	sachlich begründetes Ereignis	Untersuchung zu Studienbeginn
`Form`	Formular	Vitalparameter
`ItemGroup`	zusammengehörende Daten	Blutdruck
`Item`	einzelne Datenerhebung	Systolischer Wert

den drei Elementen `GlobalVariables`, `BasicDefinitions` und `MetaDataVersion` ist das Element `Study` vollständig.

Es folgt das Element `AdminData` (Zeile 22), das administrative Angaben enthält. Dies können Benutzer, Einrichtungen und Signaturdefinitionen sein. Ein erstes Beispiel mit Benutzern wurde bereits in Programm 3.3 auf Seite 67 zur Erläuterung von XML herangezogen. Ein komplexeres Beispiel für Benutzer, Einrichtungen und Signaturdefinitionen ist in Programm 3.23 gegeben und wird noch besprochen.

Schließlich folgt das Element `ClinicalData` (Zeile 23), das zur Aufnahme der erhobenen Daten vorgesehen ist. Da es ebenfalls umfangreichen Inhalt hat, wird es separat in Programm 3.24 gezeigt und erläutert.

ODM – Metadaten Im Detail ist ein Studienprotokoll wie folgt strukturiert: Ein Versionsstand, der als Referenz für die erhobenen Daten gilt, wird durch ein Element `MetaDataVersion` repräsentiert. Innerhalb dieses Elements befindet sich eine hierarchische Struktur der Elemente `StudyEvent`, `Form`, `ItemGroup`, `Item`. Zur Identifikation enthält jedes dieser Elemente das Attribute `OID` mit einem eindeutigen Wert. Programm 3.22 zeigt dies anhand eines Ausschnittes einer Studiendefinition in ODM. Die Struktur entsteht dadurch, dass ein definierendes Element Referenzen auf Elemente der untergeordneten Kategorie enthält. Somit ist die Definition eines Elementes unabhängig von seiner Verwendung und kann mehrfach genutzt werden. Tabelle 3.3 gibt die Elemente, ihre Bedeutung und ein Beispiel wieder.

In der XML-Darstellung wird bei der Angabe der Definitionen und Referenzen die Bezeichnung des Elementes jeweils mit dem Suffix `Def` bzw. `Ref` versehen. Programm 3.22 zeigt ein formal vollständiges ODM-Studienprotokoll, das den Blutdruck in Form des systolischen und diastolischen Wertes dokumentiert.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ODM CreationDateTime="2012-07-19T17:58:00"
3    FileOID="demo1"
4    FileType="Snapshot"
5    ODMVersion="1.3.1"
6    xmlns="http://www.cdisc.org/ns/odm/v1.3">
7    <Study OID="beispielstudie">
8      <GlobalVariables>
9        <StudyName>Beispielstudie</StudyName>
10       <StudyDescription>zu Demonstrationszwecken</
             StudyDescription>
11       <ProtocolName>Demonstration</ProtocolName>
12     </GlobalVariables>
13     <BasicDefinitions>
14       <MeasurementUnit OID="MU.MMHG" Name="mmHg">
15         <Symbol>
16           <TranslatedText xml:lang="de">mm Hg</
                 TranslatedText>
17         </Symbol>
18       </MeasurementUnit>
19     </BasicDefinitions>
20     <MetaDataVersion Name="MD.DEMO1" OID="MD.1.0"/>
21   </Study>
22   <AdminData/>
23   <ClinicalData StudyOID="beispielstudie"
         MetaDataVersionOID="MD.1.0"/>
24 </ODM>
```

Programm 3.21: Grundgerüst einer Studiendefinition in ODM.

Ausgehend vom Element `Protocol` (Zeile 2) wird die Hierarchie durch Referenzen auf untergeordnete Elemente aufgebaut. Das obligatorische Attribut `Mandatory` in der Referenz gibt an, ob das referenzierte Element zur Vollständigkeit der Studie zwangsläufig auftreten muss, oder ob es ausgelassen werden kann. Da es hier jeweils den Wert `Yes` aufweist, muss das zugehörige Element bei allen Studienteilnehmern auftreten. Das ebenfalls obligatorische Attribut `Repeating` in der Elementdefinition gibt an, ob das Element in mehreren Instanzen auftreten kann. Dies kann bspw. für eine a priori unbekannte Anzahl an Folgeuntersuchungen sinnvoll sein, bei denen stets dieselbe Fragestellung dokumentiert wird. Im Beispiel ist dies nicht nötig, der Attributwert daher `No`. Das Element `ItemDef` (Zeile 15) dient der Definition einer einzelnen Frage. Es enthält das Attribut `DataType`, mit dem der Datentyp des Messwerts angeben wird, in diesem Fall eine Ganzzahl mit 3 Stellen, wie durch das Attribut `Length` angegeben. Das Element `Question` enthält die textliche Fragestellung des zu erhebenden Messwerts. Für die Unterstützung internationaler Studien wird dieser Text pro Sprache in einem Element `TranslatedText` angegeben, dabei wird die jeweilige Sprache durch das Attribut `xlm:lang` identifiziert. Das Element `MeasurementUnitRef` verweist auf die zuvor definierte Maßeinheit. Schließlich kann der zu erhebende Wert durch das Element `Alias` mit einer semantischen Annotation versehen werden, die ein eindeutiges, maschinenlesbares Verständnis des medizinischen Konzepts bewirkt. Das Attribut `Context` enthält dazu das betreffende Begriffssystem, das Attribut `Name` enthält den Identifikator innerhalb des Begriffssystems. Falls ein Konzept durch mehrere Begriffe beschrieben wird, lässt sich das Element wiederholt verwenden. Die Definition der zweiten Frage erfolgt analog in Zeile 22.

ODM – administrative Daten Zu den administrativen Daten zählen im ODM-Format die Angabe von Personen als `User`, die Angabe von Einrichtungen als `Location` und die Definition von `Signaturen`. Ein Beispiel für die Angabe administrativer Daten ist in Programm 3.23 gegeben. Neben den hier gezeigten Kindelementen `FirstName`, `LastName` und `Email` des Elementes `User` in Zeile 2 spezifiziert ODM noch weitere optionale Eigenschaften, die Adress- und Kontaktangaben enthalten können. Eine besondere Bedeutung kommt dem Element `LocationRef` mit seinem Attribut `LocationOID` zu: Es legt mindestens eine Einrichtung, zu der der Nutzer gehört, fest und verweist dazu auf ein als `Location` festgelegtes Element, im Beispiel auf das in Zeile 8 angegebene Universitätsklinikum Münster (UKM). Die Attribute `Name` und `LocationType` geben den Namen und die Funktion der Einrichtung an, wobei `LocationType`,

```
<MetaDataVersion Name="MD.DEMO1" OID="MD.1.0">
  <Protocol>
    <StudyEventRef StudyEventOID="SE.ANAMNESE" Mandatory="Yes"/>
  </Protocol>
  <StudyEventDef OID="SE.ANAMNESE" Name="Anamnese" Type="Common" Repeating="No">
    <FormRef FormOID="F.VITALPARAMETER" Mandatory="Yes"/>
  </StudyEventDef>
  <FormDef OID="F.VITALPARAMETER" Name="Vitalparameter" Repeating="No">
    <ItemGroupRef ItemGroupOID="IG.BLUTDRUCK" Mandatory="Yes"/>
  </FormDef>
  <ItemGroupDef OID="IG.BLUTDRUCK" Name="Blutdruck" Repeating="No">
    <ItemRef ItemOID="I.SYSTOLISCH" Mandatory="Yes"/>
    <ItemRef ItemOID="I.DIASTOLISCH" Mandatory="Yes"/>
  </ItemGroupDef>
  <ItemDef OID="I.SYSTOLISCH" Name="Systolischer Wert" DataType="integer" Length="3">
    <Question>
      <TranslatedText xml:lang="de">Systolischer Wert</TranslatedText>
    </Question>
    <MeasurementUnitRef MeasurementUnitOID="MU.MMHG"/>
    <Alias Context="SNOMED" Name="399304008"/>
  </ItemDef>
  <ItemDef OID="I.DIASTOLISCH" Name="Diastolischer Wert" DataType="integer" Length="3">
    <Question>
      <TranslatedText xml:lang="de">Diastolischer Wert</TranslatedText>
    </Question>
    <MeasurementUnitRef MeasurementUnitOID="MU.MMHG"/>
    <Alias Context="SNOMED" Name="446226005"/>
  </ItemDef>
</MetaDataVersion>
```

Programm 3.22: Metadaten zur Studiendefinition in ODM.

wie im Beispiel, auf eine klinische Einrichtung hinweisen kann (`Site`), einen Sponsor (`Sponsor`), eine kommerzielle Forschungseinrichtung (`CRO`), ein Labor (`Lab`) oder eine sonstige Einrichtung (`Other`). Der Inhalt des Elements `Location` besteht aus mindestens einem Element `MetaDataVersionRef`. Es verweist auf den in einer Einrichtung verwendeten Versionsstand einer Studie und weist im Attribut `EffectiveDate` einen Datumswert auf, mit dem angegeben wird, ab wann dieser Versionsstand in der Einrichtung verwendet wird. Schließlich wird noch die Bedeutung von Signaturen durch die Element `SignatureDef` definiert. Signaturen können an erhobenen Daten angebracht werden, wobei das ODM-Format darunter sowohl Signaturen im kryptographischen Sinn, die als `Digital` bezeichnet werden, als auch nicht-kryptographische Signaturen, die `Electronic` heißen, versteht. Nicht-kryptographische Signaturen sind aus technischer Sicht einfache Zusatzinformationen zu den Daten. Da im klinischen Umfeld des Single-Source-Projektes die für kryptographische Anwendungen benötigte Infrastruktur nicht existiert, kann nur die einfachere, nichtkryptographische Signatur verwendet werden. Im Beispiel werden zwei Signaturarten definiert. Zum einen in Zeile 11 die Bestätigung eines Nutzers, dass er die signierten Daten auf Korrektheit geprüft hat. Zum anderen in Zeile 15 die Information, dass der signierende Nutzer den Transfer der Daten in das Zielsystem veranlasst hat, was nicht mit einer Prüfung der Korrektheit verbunden ist.

ODM – klinische Daten Die im Studienverlauf erfassten Daten werden innerhalb des Elements `ClinicalData` abgelegt, in Programm 3.24 findet sich ein Beispiel. Es enthält je Patient ein Element `SubjectData` (Zeile 2), das die Struktur des Studienprotokolls widerspiegelt. Das Gerüst dazu lässt sich daher aus den Metadaten der Studie generieren. Das Pseudonym des Studienteilnehmers wird durch das Attribut `SubjectKey` angegeben. Im Gegensatz zu der Definition des Studienprotokolls unter `MetaDataVersion` erfolgt der Aufbau der Hierarchie hier nicht durch Verweise, sondern durch Schachtelung der einzelnen Elemente. Die Elementnamen sind mit dem Suffix `Data` versehen und enthalten Attribute, die auf das zugehörige definierende Element verweisen. Die Bedeutung eines einzelnen `*Data`-Elements ergibt sich aus den in der Hierarchie übergeordneten Elementen. So könnte ein Studienprotokoll festlegen, dass der Blutdruck nicht nur in der anfänglichen, sondern auch in weiteren Untersuchungen dokumentiert wird. Dann würden mehrere Elemente `ItemGroupData` mit dem Attributwert `IG.BLUTDRUCK` auftreten, durch die übergeordneten Elemente könnte aber entschieden werden, zu welchem Formular und Stu-

```
<AdminData StudyOID="beispielstudie">
  <User OID="musterfrau" UserType="Investigator">
    <FirstName>Petra</FirstName>
    <LastName>Musterfrau</LastName>
    <Email>petra@example.org</Email>
    <LocationRef LocationOID="UKM"/>
  </User>
  <Location OID="UKM" Name="Universitätsklinikum Münster"
      LocationType="Site">
    <MetaDataVersionRef EffectiveDate="2013-01-01"
        StudyOID="beispielstudie" MetaDataVersionOID="MD
        .1.0"/>
  </Location>
  <SignatureDef OID="SG.APPROVAL" Methodology="Electronic
      ">
    <Meaning>Bestätigt</Meaning>
    <LegalReason>Der Unterzeichner bestätigt die
        Korrektheit der Daten</LegalReason>
  </SignatureDef>
  <SignatureDef OID="SG.TRANSFER" Methodology="Electronic
      ">
    <Meaning>Datenübertragung</Meaning>
    <LegalReason>Der Unterzeichner hat den automatischen
        Datentransfer in das Zielsystem veranlasst, aber
        die übertragenen Daten nicht geprüft</LegalReason
        >
  </SignatureDef>
</AdminData>
```

Programm 3.23: Angabe administrativer Daten in ODM.

```
1  <ClinicalData StudyOID="beispielstudie"
       MetaDataVersionOID="MD.1.0">
2    <SubjectData SubjectKey="PSEUDO.1">
3      <InvestigatorRef UserOID="musterfrau"/>
4      <StudyEventData StudyEventOID="SE.ANAMNESE">
5        <FormData FormOID="F.VITALPARAMETER">
6          <ItemGroupData ItemGroupOID="IG.BLUTDRUCK">
7            <ItemData ItemOID="I.SYSTOLISCH" Value="120"/>
8            <ItemData ItemOID="I.DIASTOLISCH" Value="80"/>
9          </ItemGroupData>
10         <Signature>
11           <UserRef UserOID="musterfrau"/>
12           <LocationRef LocationOID="UKM"/>
13           <SignatureRef SignatureOID="SG.APPROVAL"/>
14           <DateTimeStamp>2013-02-01T09:44:20</
                 DateTimeStamp>
15         </Signature>
16       </FormData>
17     </StudyEventData>
18   </SubjectData>
19 </ClinicalData>
```

Programm 3.24: Klinische Daten in ODM.

dienereignis sie gehören. Das Element `ItemData` enthält das Attribut `Value`, das den entsprechenden Messwert aufnimmt. Signaturen können auf allen Ebenen zwischen `ClinicalData` und `ItemData` auftreten und beziehen sich jeweils auf das Element, in dem sie enthalten sind, und dessen Kindelemente. Im Beispiel existiert durch das Element `Signature` in Zeile 10 eine Signatur auf Ebene des Formulars. Es enthält immer Verweise auf die unter `AdminData` angelegten Elemente für den unterzeichnenden Nutzer (`UserRef`), die Einrichtung (`LocationRef`) und die Signaturdefinition (`SignatureRef`). Zudem wird der Zeitpunkt der Signatur durch das Element `DateTimeStamp` (Zeile 14) angegeben.

Falls in den Studienmetadaten eines oder mehrere der Elemente `StudyEventDef`, `FormDef` oder `ItemGroupDef` mit dem Attribut `Repeating="Yes"` als wiederholbar auszuführende Dokumentation gekennzeichnet sind, so müssen die zugehörigen `*Data`-Elemente ein zusätzliches Attribut erhalten, um eine einzelne Instanz identifizieren zu können. Dazu sind die Attribute `StudyEvent-`

```
1    <FormDef OID="F.VITALPARAMETER" Name="Vitalparameter"
         Repeating="Yes">
2      <ItemGroupRef ItemGroupOID="IG.BLUTDRUCK" Mandatory="
           Yes"/>
3    </FormDef>
4    <ItemGroupDef OID="IG.BLUTDRUCK" Name="Blutdruck"
         Repeating="No">
5      <ItemRef ItemOID="I.SYSTOLISCH" Mandatory="Yes"/>
6      <ItemRef ItemOID="I.DIASTOLISCH" Mandatory="Yes"/>
7    </ItemGroupDef>
```

Programm 3.25: Variation der Formulardefinition aus Programm 3.22. Das Formular Vitalparameter kann mehrfach ausgefüllt werden.

```
1  <FormData FormOID="F.VITALPARAMETER" FormRepeatKey="1">
2    <ItemGroupData ItemGroupOID="IG.BLUTDRUCK">
3      <ItemData ItemOID="I.SYSTOLISCH" Value="120"/>
4      <ItemData ItemOID="I.DIASTOLISCH" Value="80"/>
5    </ItemGroupData>
6  </FormData>
7  ...
8  <FormData FormOID="F.VITALPARAMETER" FormRepeatKey="2">
9    <ItemGroupData ItemGroupOID="IG.BLUTDRUCK">
10     <ItemData ItemOID="I.SYSTOLISCH" Value="125"/>
11     <ItemData ItemOID="I.DIASTOLISCH" Value="83"/>
12   </ItemGroupData>
13 </FormData>
```

Programm 3.26: `FormData`-Elemente des wiederholt ausfüllbaren Formulars Vitalparameter aus Programm 3.25.

`RepeatKey`, `FormRepeatKey` und `ItemGroupRepeatKey` vorgesehen. In Programm 3.25 ist eine Variante der Formulardefinition aus Programm 3.22 gezeigt, bei der das Formular Vitalparameter wiederholt dokumentiert werden kann. Dazugehörige `FormData`-Elemente sind in Programm 3.26 zu finden. Sie weisen jeweils das Attribut `FormRepeatKey` mit einem Schlüsselwert auf. Ein möglicher Anwendungsfall dafür könnte sein, dass die Vitalparameter des Patienten zu Beginn und zum Abschluss des Studienereignisses gemessen werden soll.

4 Entwicklungsmethode

Im Rahmen dieses Kapitels wird zunächst der in dieser Arbeit verfolgte Lösungsansatz für die in Abschnitt 2.3 identifizierten Brüche der traditionellen Schichtenarchitektur unter Verwendung der in Kapitel 3 beschriebenen Techniken skizziert und die Grundlagen der hier vorgestellten XML-basierten Architektur werden aufgezeigt. Sodann wird eine Softwareentwicklungsmethode eingeführt, die zu der hier vorgestellten XML-basierten Architektur passt. Diese folgt den Phasen Analyse, Entwurf, Implementierung sowie Einführung und Nutzung, in denen jeweils geeignete Werkzeuge zum Einsatz kommen. In der Analysephase werden die zugrunde liegenden Prozesse identifiziert und modelliert. Ausgehend davon werden die dabei benötigten Daten und organisatorischen Aspekte bestimmt. In der Entwurfsphase werden die Prozesse weiter detailliert, so dass die benötigten Funktionen deutlich werden. Hierbei müssen Anforderungen an die Funktionen erkannt werden, um entscheiden zu können, ob diese durch die Software selbst implementiert werden sollen, oder als Services eingebunden werden können. Services können hierbei sowohl organisationsintern als auch durch Dritte bereitgestellt werden. Für Funktionen, die als Services bezogen werden sollen, werden sodann verfügbare Anbieter gesucht. Davon ausgehend, welche Funktionen durch die Anwendung bereitgestellt oder verwendet werden sollen, werden die Schnittstellen für Maschinen und menschliche Nutzer definiert. Sodann wird ein zu den Anforderungen des Datenmodells passendes XML Schema gesucht. Je nach Anforderung bieten sich hierzu bestehende Standards oder Eigenentwicklungen an. Die Ausführungen zur Implementierungsphase geben Hinweise zur Umsetzung der Software. Das Kapitel schließt mit einer Betrachtung der Einführungs- und Nutzungsphase.

4.1 Architekturprinzip

Ein zentrales Ziel der hier vorgestellten XML-basierten Architektur ist die Vermeidung von Brüchen im Schichtenmodell. Grundsätzlich soll ein Schichtenmodell beibehalten werden, da sich die Aufteilung der Funktionen in Schichten als nützlich etabliert hat [SD00]. Vermieden werden sollen hingegen die Paradigmenbrüche zwischen den häufig unterschiedlichen Konzepten der einzelnen Schichten, insbesondere der schwerwiegende objektrelationale Impedance Mis-

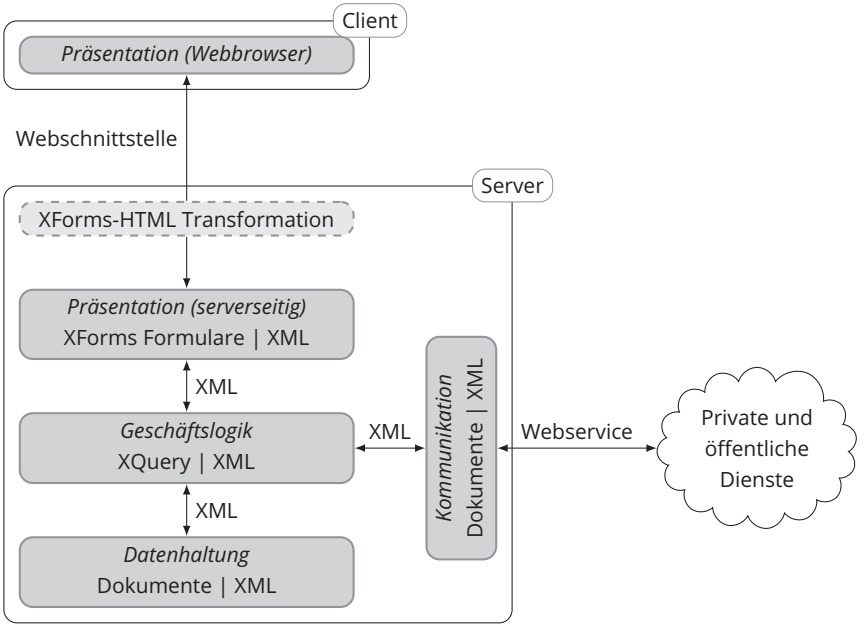


Abbildung 4.1: XML-basierte Architektur, die auf Paradigmenbrüche verzichtet. Da gängige Browser XForms nicht interpretieren können, wird einzig ein automatisierter XForms-HTML Konverter benötigt.

match zwischen Persistenzschicht und Geschäftslogik. Auch die Anbindung von Präsentationsschicht und die Einbindung von Diensten sollen möglichst wenig manuelles Mapping erfordern. Dazu ist die Wahl der Paradigmen und Techniken der einzelnen Schichten nicht mehr nur am vermeintlichen Standard auszurichten, sondern es ist auch auf die Interoperabilität zu achten. In diesem Zusammenhang bedeutet dies, dass die Übergänge zwischen den Schichten möglichst ohne den Einsatz von Mapping-Software bewältigt werden sollen.

Wie im vorherigen Kapitel gezeigt, bieten der W3C-Standard XML und darauf basierende Techniken Lösungen für Persistenz, Geschäftslogik, Präsentation und Kommunikation. Ersetzt man die traditionellerweise in der Schichtenarchitektur verwendeten Paradigmen durch entsprechend geeignete XML-Standards, so ergibt sich die in Abbildung 4.1 gezeigte Architektur: Anstelle der relationalen Speicherung in den Tabellen eines relationalen DBMS erfolgt die Speiche-

Tabelle 4.1: Eigenschaften der Schichten bei traditioneller und XML-basierter Architektur.

Schicht	Traditionelle Architektur	XML-Architektur
Persistenz	relationales DBMS	XML-DBMS
Geschäftslogik	Objektorientierte Programmierung	XQuery
Kommunikation	XML und Webservices	XML und Webservices
Präsentation	HTML via OO-Framework	HTML via XForms

rung von XML-Dokumenten in einem XML-DBMS. Die interne Repräsentation der XML-Daten durch das DBMS wird dabei als Blackbox angesehen, da die hier behandelte XML-basierte Architektur konzeptionell auf die Schnittstellen gerichtet ist. Das DBMS kann intern beliebige Optimierungen vornehmen, von denen hier abstrahiert wird. Die Kommunikation zwischen dieser Schicht und der Geschäftslogik erfolgt in XQuery, also einer für XML optimierten Sprache. Die dabei ausgetauschten Daten sind in XML codiert. Die Implementierung der Geschäftslogik erfolgt ebenfalls in XQuery. Somit muss keine Transformation der Daten vorgenommen werden, da das aus der Persistenzschicht bekannte Datenmodell weiter verwendet werden kann. Bei der Einbindung von SOAP Webservices kann ebenfalls auf Transformation des Formats verzichtet werden, da diese meist[1] XML-Nachrichten austauschen. Bei der Einbindung der Präsentationsschicht, speziell der Gestaltung von Formularen, kommt mit XForms ein Standard zum Einsatz, dessen Datenmodell auch auf XML basiert. Da für XForms keine stabilen nativen, clientseitigen Interpreter vorliegen, muss allerdings weiterhin eine Transformation zu HTML-Formularen und ggf. JavaScript erfolgen. Die verfügbaren serverseitigen XForms Interpreter generieren aber den notwendigen Code, sodass konzeptionell von dieser Transformation abstrahiert werden kann. Somit kann durch den gesamten Architekturstack das XML-basierte Datenmodell verwendet werden. Tabelle 4.1 stellt die unterschiedlichen Ansätze gegenüber. In den folgenden Abschnitten werden die Eigenschaften im Entwicklungsprozess bei Verwendung dieser XML-basierten Architektur herausgearbeitet. Angenommen wird eine Entwicklung nach agilem Vorgehen, ohne dass die XML-Architektur darauf beschränkt ist.

[1] Im Januar 2014 verwenden 2045 der 2134 von ProgrammableWeb.com [30] aufgelisteten SOAP-Dienste XML als Datenformat.

4.2 Analyse

Die Analysephase abstrahiert von der technischen Umsetzung und hat zum Ziel, das Sollkonzept zu erstellen. Dazu werden die fachlichen Anforderungen analysiert und in Form von Geschäftsprozessen detailliert dargestellt.

Der vorgeschlagene Entwicklungsprozess für die hier behandelte XML-basierte Architektur setzt dabei auf den Einsatz vorhandener Werkzeuge wie XML-Netze und UML Klassendiagramme. Zunächst werden die Abläufe, die das Softwaresystem unterstützen soll, modelliert. Sodann wird ein Objektmodell entwickelt, das immaterielle und materielle Geschäftsobjekte enthält, an denen gemäß des Ablaufmodells Aktivitäten verrichtet werden und die den im Ablaufmodell festgelegten Pfaden folgen. Schließlich wird die Organisationsstruktur modelliert.

4.2.1 Identifikation des zugrunde liegenden Prozesses

Die Aktivitäten um das Management von Geschäftsprozessen verfolgen nach [Wes12] die folgenden Ziele:

- Besseres Verständnis der Abläufe im Unternehmen
- Verständigung mit Stakeholdern
- Flexibilität als Fähigkeit zur Veränderung
- Aufbau eines Wissensspeichers über das Unternehmen
- Evolutionäre Verbesserung der Prozesse
- Grundlage für die Umsetzung in Software

Für die Analysephase im Rahmen des Softwareentwurfs ist der letztgenannte Punkt relevant. Die „Vision“ ist ein präzise spezifiziertes Verhältnis zwischen Geschäftsprozessen und deren Realisierung in Software [Wes12].

Aus diesem Grund erscheint eine Modellierung, die mit der späteren Umsetzung der Software harmoniert, erstrebenswert. Dies ist kein Widerspruch zu der Forderung, dass die Ergebnisse der Analysephase unabhängig von der technischen Umsetzung sein sollen, da die Umsetzung erst in der nachfolgenden Entwurfsphase spezifiziert wird. Zueinander passende Methoden erleichtern aber den Übergang.

Für die Modellierung von Prozessen sind in der Literatur unterschiedliche Techniken bekannt: Die bereits in Abschnitt 3.4.1 beschriebenen Petri-Netze

und darauf aufbauende Erweiterungen, insbesondere XML-Netze. Die *Aktivitätsdiagramme* der UML, deren Tokenkonzept Ähnlichkeit zu Petri-Netzen aufweist [HKKR05]. *Business Process Markup Language (BPMN)* ist eine Notationssprache, mit der Geschäftsprozesse graphisch modelliert werden können. Die Sprachspezifikation [Obj11] bietet dazu umfangreiche Elemente und eine standardisierte XML-Serialisierung [FR12]. Eine weitere Möglichkeit zur Modellierung von Geschäftsprozessen sind *Ereignisgesteuerte Prozessketten (EPKs)*, die im Konzept Architektur integrierter Informationssysteme (ARIS) zum Einsatz kommen [Sch01]. Für die Analysephase einer Softwareentwicklung, die auf der XML-basierten Architektur aufbaut, werden in der vorliegenden Arbeit XML-Netze verwendet. Sie integrieren als einzige der genannten Methoden die Beschreibung des Datenmodells mit XML Schemata und unterstützen mit Filterschemata dazu passende Verarbeitungsanweisungen. XML-Netze lassen sich unabhängig von der späteren Implementierung zur Prozessmodellierung einsetzen. Besonders naheliegend ist ihre Verwendung im Rahmen einer XML-basierten Architektur: Während des Analyse- und Entwurfsprozesses in Form von XML Schema dokumentierte Modelle können in der Implementierung verwendet werden. XML-Netze lassen es zu, während der Analyse Abläufe auf einem höheren Abstraktionsgrad zu beschreiben und in der Entwurfsphase durch Verfeinerung und ergänzende Beschreibung zu detaillieren.

Die Literatur zur Modellierung von Geschäftsprozessen kennt unterschiedliche Vorgehensmodelle, die im Kern ähnlich funktionieren, aber unterschiedliche Methoden und Tools nutzen [SVOK11; BKR12; Fis12; Wes12; Gad13; Jos08]. Hier kommt mit dem Prinzip der Funktionsdekomposition [Wes12] eine Topdown-Methode zum Einsatz. Was auf höherem Abstraktionsniveau als einzelne Aktivität modelliert ist, wird auf detaillierterem Niveau in seinen Bestandteilen dargestellt. Dazu wird das Prozessmodell in Abstraktionsebenen eingeteilt. Ebene 0 ist die Ebene mit dem höchsten Abstraktionsgrad, die folgenden Ebenen 1 bis n verfeinern Teile davon sukzessiv.

Abbildung 4.2 zeigt das Prinzip anhand der Verfeinerung eines XML-Netzes, das einen vereinfachten Prozess, den ein Patient in einem Krankenhaus durchläuft, darstellt. Auf Ebene 0 ist zu sehen, dass für den Patienten nach der Aufnahme die Dokumentationsakte in Form des Electronic Health Record (EHR) und die Dokumentation für eine klinische Studie angelegt werden können, woraufhin Behandlung und Studie durchführbar sind. Auf Ebene 1 wird die Studiendurchführung detailliert. Sie besteht aus einer Untersuchung und der Dokumentation des Befunds. Die Dokumentation wird auf Ebene 2 verfeinert, sie besteht aus dem Aufruf des entsprechenden Dokumentationsformulars sowie dessen Bear-

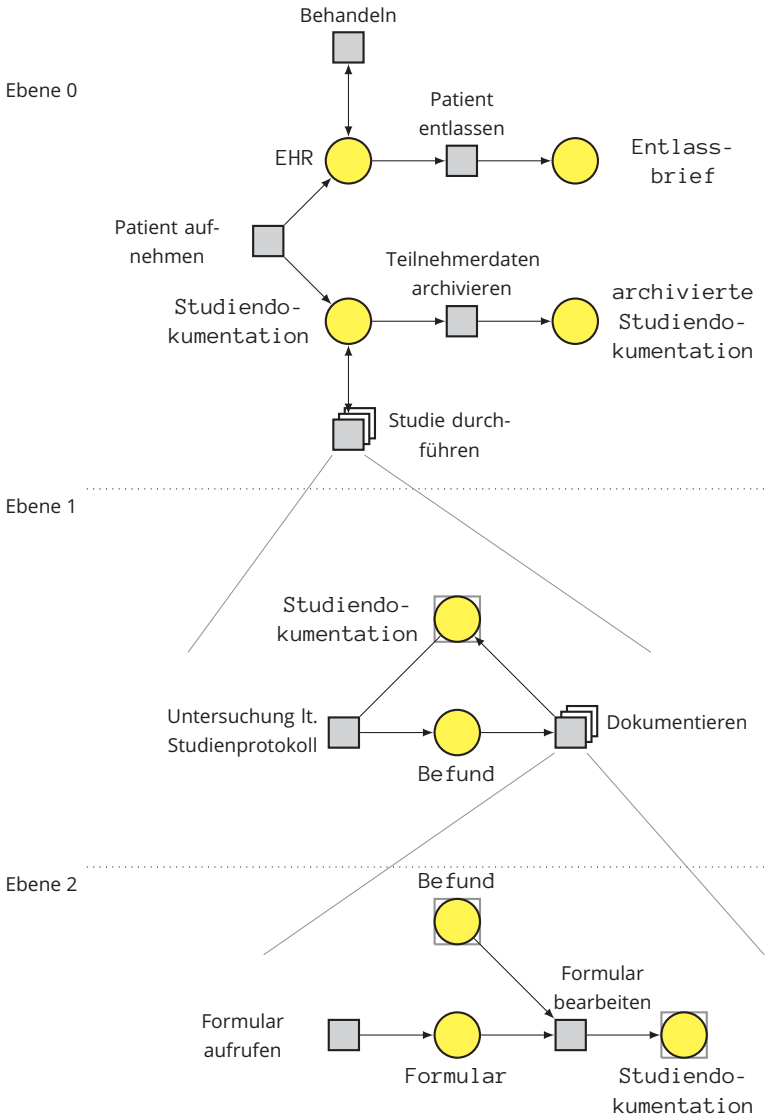


Abbildung 4.2: Funktionsdekomposition im Ebenenmodell. Das Beispiel zeigt die Verfeinerung des vereinfachten Prozesses, den ein Patient im Krankenhaus durchläuft, wenn er neben seiner Behandlung noch Teilnehmer einer Studie ist.

beitung und Speicherung in der Studiendokumentation.

In der Literatur finden sich zur Modellierung höherer Ebenen informelle Darstellungen [Jos08] oder Darstellungen, die sich an der Aufbauorganisation und deren Funktionen orientieren [Wes12], und dazu je nach Ebene unterschiedliche Modellierungstechniken verwenden. Die vorliegende Arbeit schlägt vor, auf den Ablauf der Prozesse zu fokussieren und bereits auf Ebene 0 die Technik der XML-Netze zu verwenden.

Die Fragen nach dem richtigen Umfang eines Modells und einem angemessenen Detailgrad eines einzelnen Prozesses können hierbei nicht pauschal beantwortet werden. Es liegt nahe, die Grundsätze ordnungsmäßiger Modellierung (GOM) [BPV12] heranzuziehen. Diese stellen Kriterien dar, die zu hoher Modellqualität führen:

- „Grundsatz der Richtigkeit
- Grundsatz der Relevanz
- Grundsatz der Wirtschaftlichkeit
- Grundsatz der Klarheit
- Grundsatz der Vergleichbarkeit
- Grundsatz des systematischen Aufbaus“ [BPV12]

Der Grundsatz der *Richtigkeit* umfasst dabei sowohl syntaktische Richtigkeit, also die formal korrekte Anwendung der Modellierungstechnik, also auch semantische Richtigkeit, d. h. einen Konsens der „Gutwilligen und Sachkundigen“ [KL96] zit. n. [BPV12] über die Modellelemente. Unter allen Modellnutzern sollte also Einigkeit über die Bedeutung der verwendeten Elemente hergestellt werden. Der Grundsatz der *Relevanz* fordert, dass sämtliche für den Modellierungszweck relevanten Sachverhalte abgebildet und keine nichtrelevanten Modellelemente vorhanden sind. Genau die Sachverhalte, die zum Verständnis der Prozesse benötigt werden, sind also abzubilden. Der Grundsatz der *Wirtschaftlichkeit* postuliert, ein Modell nur soweit zu verfeinern, wie der Nutzen der zusätzlichen Detaillierung die Kosten übersteigt. Für die Prozessmodellierung im Rahmen der Analysephase bedeutet dies, dass das Modell soweit verfeinert wird, bis fachliche Aktivitäten so detailliert dargestellt sind, dass der folgende technische Entwurf darauf aufbauen kann. Weitere Details, die für die Analyse nicht nötig sind, werden erst in der Entwurfsphase ergänzt. Der Grundsatz der *Klarheit* gibt vor, dass ein Modell verständlich sein muss. Dazu gehören im

Bereich des Layouts die analoge Darstellung strukturgleicher Sachverhalte, das Vermeiden von Überschneidungen der Modellelemente und das Einhalten von Gestaltungskonventionen wie bspw. dass die Flussrichtung in einem Prozessdiagramm gemäß der Leserichtung grundsätzlich von links nach rechts verlaufen sollte. Die Modellgröße muss zu dem zur Betrachtung vorgesehenen Medium passen, in diesem Fall sollte die Größe so gewählt werden, dass das Modell auf einem üblichen Bildschirm dargestellt werden kann. Größere Modelle müssen entsprechend hierarchisiert werden. Der Grundsatz der *Vergleichbarkeit* umfasst einerseits die Forderung, dass gleiche Sachverhalte in der Realität zu identischen Modellen führen müssen und andererseits, dass mit unterschiedlichen Techniken erstellte Modelle ineinander überführbar sein müssen, um sie vergleichen zu können. Für die Prozessmodellierung während der Analysephase kann dies von Bedeutung sein, falls bereits Modelle vorhanden sind, die weiter genutzt werden sollen. Eine Anwendung dafür wäre die Existenz von Prozessmodellen, die mittels BPMN erstellt wurden: Für BPMN ist eine weitestgehend formale Transformation zu Petri-Netzen möglich [DDO08]. Der Grundsatz des *systematischen Aufbaus* erfordert, dass ein Gesamtmodell, das aus unterschiedlichen Sichten besteht, aus allen Sichten betrachtet konsistent aufgebaut sein muss. Elemente, die in mehreren Sichten auftreten, müssen mit identischer Bedeutung verwendet werden. So müssen die im Prozessmodell verwendeten Datenobjekte bspw. im Datenmodell konsistent modelliert werden.

4.2.2 Identifikation der Datenobjekte

Nach der Prozessanalyse werden die Datenobjekte, die von Aktivitäten produziert, konsumiert oder gelesen werden, analysiert, um in der Entwurfsphase als XML Schema-Dokumente spezifiziert zu werden. Während für die Modellierung von Datenstrukturen nach dem relationalen Paradigma mit ERM eine erprobte Technik zur Verfügung steht, existiert keine allgemein anerkannte konzeptionelle Modellierungstechnik für XML Schema-Dokumente. In Abschnitt 3.1.4 wurden verschiedene Ansätze zur Modellierung von XML Schema vorgestellt, von denen sich aber keiner als Standardvorgehen in der Praxis durchsetzen konnte.

In Abwesenheit eines Standardvorgehens zur Modellierung von XML Schema werden die Geschäftsobjekte im Rahmen der Analysephase der XML-basierten Architektur aufgrund der Bekanntheit und breiten Toolunterstützung mittels UML Klassendiagrammen entwickelt. Die XML Schema-spezifischen Einschränkungen des UML-Modells und die weitere Verfeinerung können damit im Rahmen des Entwurfs vorgenommen werden.

Abbildung 4.3 zeigt ein UML-Modell zur Analyse der Datenobjekte. Die analysierten Datenobjekte werden für die Erweiterung eines herkömmlichen Krankenhausinformationssystems (KIS) benötigt, das mit einem Electronic Data Capture (EDC)-System integriert werden soll. Da der Aufruf des EDC aus dem KIS erfolgen soll, müssen im Datenmodell des KIS die dazu nötigen Informationen hinterlegt werden. Die Elemente *User* und *Patient* sind bereits im KIS vorhanden. Mit den vorhandenen Patientendaten müssen Informationen über die Teilnahme des Patienten an einer oder mehreren Studien verknüpfbar sein. Dazu wird das Element *Studienteilnehmer* verwendet. Für jede dieser Studienteilnahmen muss ebenfalls gespeichert werden, welche Personen eine *Zugriffsberechtigung* auf die Studiendokumentation des Teilnehmers erhalten sollen. Für die Verknüpfung zu dem EDC-System und den Zugriff auf unterschiedliche Dokumentationsformulare werden die *Dokumentationselemente* abgebildet. Optional können mit einem Dokumentationselement ein *Zeitplan* sowie ein *Status* für die Bearbeitung verknüpft werden.

4.2.3 Organisationsstruktur

Die Organisationsstruktur eines Unternehmens ist die Grundlage für die Zuordnung von Prozessschritten zu ausführenden Organisationseinheiten. Sie ist die entscheidende Dokumentation für die Berechtigungsverwaltung der Software. Für die Modellierung von Organisationsstrukturen finden häufig Organigramme Anwendung. Allerdings hat sich keine standardisierte Darstellungsweise etabliert.

Ein eher an betriebswirtschaftlichen Erfordernissen orientiertes Vorgehen findet sich in [Kug00]. Es enthält differenzierte Elemente zur Beschreibung von Organisationseinheiten, Gremien, Stellentypen, Stellen und Rollen sowie mehrere semantisch unterschiedliche Beziehungsarten. Zur Softwareentwicklung wird eine derart detaillierte Beschreibung im Allgemeinen nicht benötigt.

Im Rahmen des ARIS-Konzepts wird eine sehr einfache Organigrammdarstellung mit ungerichteten Kanten verwendet [Sch01], bei der sich die hierarchische Position einer Organisationseinheit nur durch die physische Position des repräsentierenden Elements im Modell ergibt. Da die Abbildung von Hierarchien aber das wesentliche Merkmal eines Organigramms darstellt, erscheint eine explizite Unterstützung dieser durch ein Modellelement wünschenswert.

In [SVOK11] findet sich ein offenbar an UML angelehntes Organisationsmodell. Hierbei existiert lediglich ein an Klassen erinnerndes Modellelement zur Beschreibung von Organisationseinheiten. Die hierarchischen Beziehungen

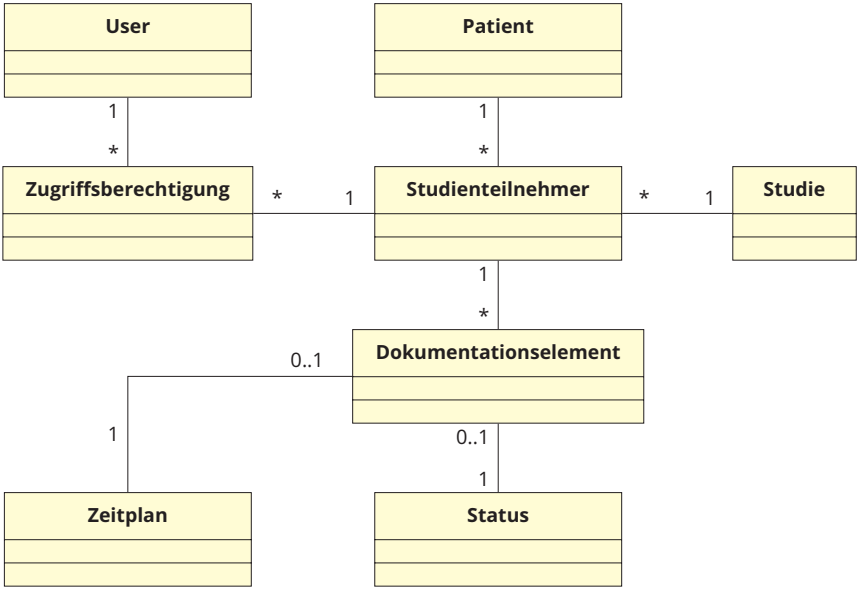


Abbildung 4.3: Analyse der Datenobjekte in UML. Die zusätzlich zu User und Patient gezeigten Datenobjekte stellen die für die Anbindung eines Studiendokumentationssystems in einem Krankenhausinformationssystem notwendigen Erweiterungen dar.

werden durch Assoziationskanten dargestellt, wobei die Raute an der hierarchisch übergeordneten Einheit angebracht ist.

Im Folgenden wird diese Modellierungsweise übernommen, allerdings werden die Standardelemente der UML verwendet. Abbildung 4.4 illustriert ein derartiges Organigramm. Exemplarisch wird die Hierarchie der Rollen eines Studiendokumentationssystems gezeigt. Die Rolle des Systemadministrators ist die hierarchisch höchste und erlaubt die Nutzung aller Systemfunktionen, darunter ist die des Studienadministrators angesiedelt, der alle Rechte für eine einzelne Studie besitzt. Der Standortadministrator verfügt über diese Rechte für einen Standort innerhalb einer Studie. Die Rollen Standortauswerter und Datensammler verfügen nur über die zum Auswerten bzw. Erheben von Daten an ihrem Standort nötigen Rechte.

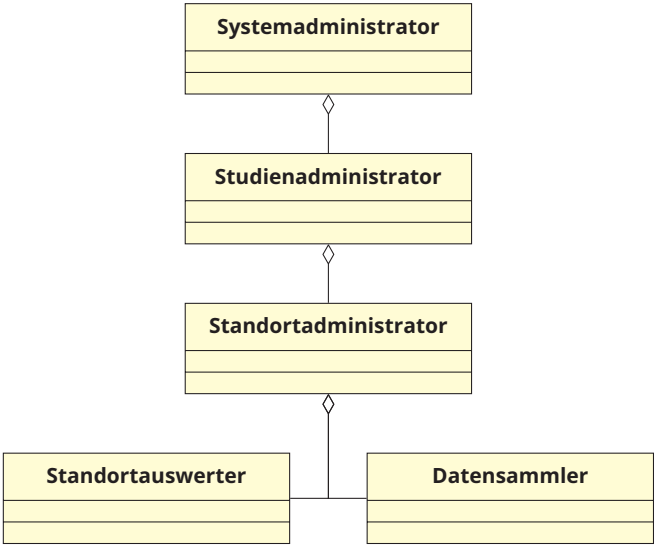


Abbildung 4.4: Organigrammdarstellung mittels UML.

4.3 Entwurf

Während der Entwurfsphase wird das in der Analyse erstellte Geschäftsprozessmodell für die technische Umsetzung vorbereitet. Dazu werden die Prozesse insoweit verfeinert, als dass nicht mehr nur angegeben wird, was die Funktionen tun, sondern auch wie es geschehen soll. Die Beschreibung der Objekte wird um Details ergänzt und entsprechende XML Schemata werden identifiziert, falls standardisiert vorhanden, oder anderenfalls neu erstellt. Das Organigramm wird um eine Beschreibung der Funktionen, die durch die einzelnen Rollen ausgeführt werden, ergänzt.

4.3.1 Prozesse

Der auf dem Prozessmodell basierende Teil der Entwurfsphase behandelt die Spezifizierung der darin verwendeten Funktionen. Dazu werden das Prozessmodell verfeinert, Kriterien für das Sourcing von Funktionen bestimmt, Anforderungen an die Funktionen festgehalten und die Schnittstellen für maschinelle und menschliche Nutzer der Funktionen festgelegt.

Verfeinerung der Prozessmodelle

Die während der Analysephase identifizierten Geschäftsprozesse werden nun soweit detailliert, dass die dafür benötigten Funktionen implementiert werden können. Dazu werden die XML-Netze entlang der Kanten mit Verarbeitungsinformationen versehen. Der dazu in [LO02] für XML-Netze vorgesehene Ansatz verwendet „Filterschemata" mit der Anfrage- und Manipulationssprache „XManiLa". XManiLa adaptiert das Prinzip *Query-By-Example* [LO03]. Diese Anfragesprache wurde im Jahr 1975 als einfache Abfragemöglichkeit für relationale DBMS vorgestellt [Zlo75]. Nutzern ohne Hintergrundwissen über Datenbanktechnik sollte es ermöglicht werden, eigenständig Anfragen formulieren zu können. Dazu bedient der Benutzer einen speziellen Editor, in dem er die Eigenschaften der gesuchten Datenbankeinträge beispielhaft vorgibt. Das DBMS selektiert dann die Einträge mit passenden Eigenschaften. XManiLa verwendet eine grafische, auf GXSL basierende Notation zur Selektion der (Teil-)Dokumente, die gelesen oder manipuliert werden sollen. Diese scheint für einfache Anfragen angemessen, allerdings führt die Modellierung komplexer Anfragen zu komplizierten Graphen. XManiLa ist für den Umgang mit XML Dokumenten entworfen, weitere Eigenschaften von XQuery, wie z. B. Definition und Aufruf von Funktionen, die ggf. Seiteneffekte haben, lassen sich damit nicht abbilden. Ebenfalls damit nicht modellierbar sind XQuery-Konstrukte wie Sequenzen, Sortierausdrücke und Join-Operationen. Die beschränkte Eignung von XManiLa zur Modellierung von Datenmanipulationsoperationen in XML-Netzen wird auch in [LOZ11] festgestellt und eine XSLT-basierte Alternative vorgeschlagen. XManiLa ist also für den Entwurf der Programmlogik mit XQuery nicht ausreichend. In [SVOK11] wird auf Pseudocode zur Beschreibung der Filterschemata zurückgegriffen, in [Kar12] wird auf die Möglichkeit der Darstellung von Filterschemata mittels XQuery hingewiesen. Da XQuery und XSLT als ausführbare Programmiersprachen konzipiert sind, darf die Eignung als Modellierungssprache bezweifelt werden. Für den Entwurf der XML-basierten Architektur wird daher auch auf Pseudocode zurückgegriffen, falls im Rahmen der Entwurfsphase Funktionen verdeutlicht werden müssen. Abbildung 4.5 detailliert das Prozessmodell aus Abbildung 4.2 mit Ergänzungen zur Funktionsweise der einzelnen Prozessschritte.

Sourcing

Während etablierte Ansätze zur Softwareentwicklung wie z. B. das V-Modell XT die zu erstellende Software als mehr oder weniger in sich abgeschlossenes Werk

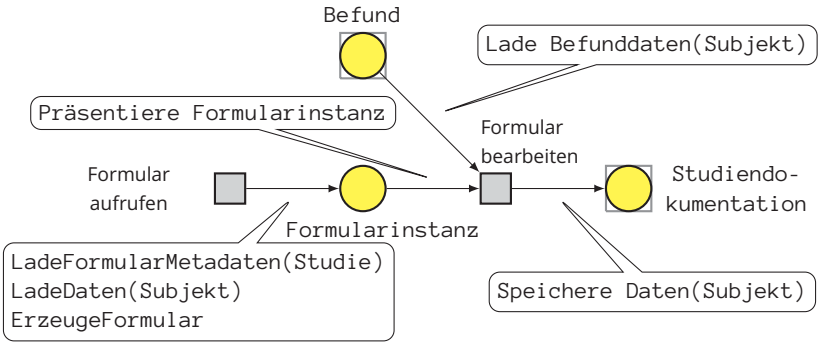


Abbildung 4.5: Weitere Spezifikation des Subprozesses Dokumentieren des Ebene 2-Modells aus Abbildung 4.2 im Rahmen des Entwurfs.

verstehen, so sehen jüngere Entwicklungen wie SOA [MLM+06; Jos08], Web-orientierte Architekturen (WOA) [Thi11] und Cloud-Computing [VHH12; Has12] die Einbindung von Services, die nicht unter der direkten Kontrolle des Softwareanwenders stehen, im Sinne eines verteilten Systems vor. Auch bei der Entwicklung von Software mit der XML-basierten Architektur stellt sich die Frage, ob Funktionen selbst entwickelt werden sollen oder ob es bereits Services gibt, die eingebunden werden können. Dabei sind sowohl unternehmensinterne als auch externe Serviceanbieter zu berücksichtigen. Gründe für den Einsatz externer Serviceanbieter, also das IT Outsourcing, sind die Erbringung besserer Leistung als im Eigenbetrieb, der Wunsch, schneller an technischen Verbesserungen partizipieren zu können und erwartete Kostensenkungen [GGL10]. Um zu entscheiden, ob Funktionen per Service genutzt oder durch die Software implementiert werden sollen, muss betrachtet werden, ob die Funktion für die Nutzung per Service geeignet ist, und wenn ja, ob sie durch externe oder bereits vorhandene oder noch zu implementierende interne Services realisiert werden kann. Andernfalls ist eine Realisierung als Komponente der Software nötig.

Abbildung 4.6 gibt den Entscheidungsprozess wieder: Wenn eine Funktion grundsätzlich für die Auslagerung als Webservice geeignet ist, wird entschieden, ob dieser intern betrieben werden soll, oder ob ein externer Service zum Einsatz kommen kann. Falls ein externer Servicebezug möglich ist und es geeignete Angebote gibt, wird ein externer Webservice eingebunden. Falls nur unternehmensinterner Servicebezug möglich ist, muss geprüft werden, ob ein derartiger Service bereits vorhanden ist oder ggf. erst erstellt und schließlich

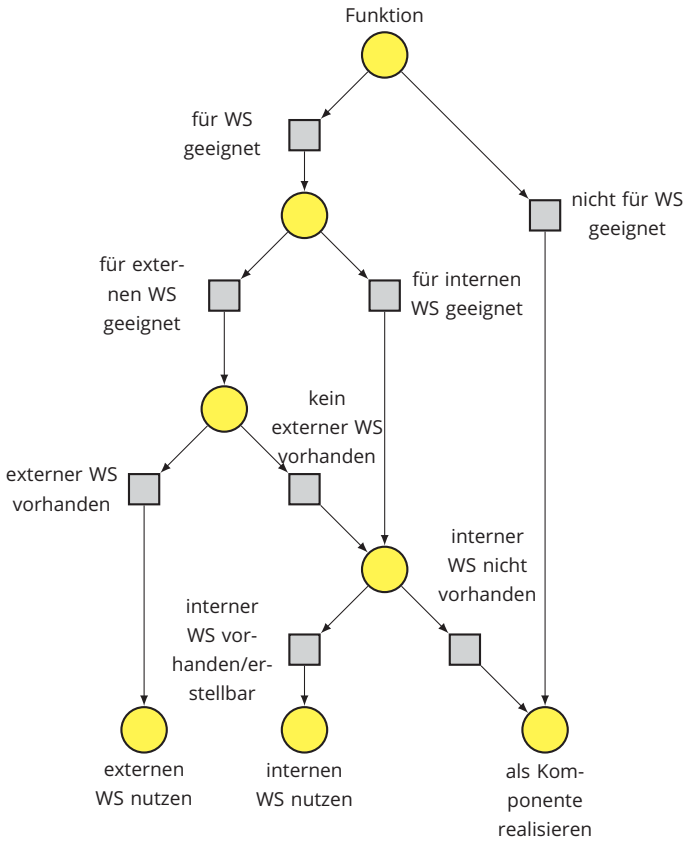


Abbildung 4.6: Entscheidungsprozess für ein Sourcing von Funktionen.

genutzt werden kann. Falls dies nicht möglich ist, muss ebenso wie bei Funktionen, die nicht für Webservices geeignet sind, eine Realisierung als Komponente der zu erstellenden Software erfolgen. Für die in diesem Prozess zu treffenden Entscheidungen sind je nach Anwendungsdomäne der Software und Relevanz der Funktionen unterschiedliche Kriterien heranzuziehen. Zunächst muss geklärt werden, ob sich eine Funktion aus technischer Sicht für die Auslagerung in einen Webservice eignet. Dabei stellt die derzeit gängige, nachrichtenbasierte Interaktion mit Webservices eine Limitation dar. Streambasier-

te Interaktionen werden derzeit noch entwickelt [LR13; RTS13] und sind noch nicht standardisiert. Daher eignen sich Webservices momentan nicht für Daten, die als Streams übertragen werden, wie z. B. Audio- und Videokommunikationsdaten. Als weitere Schwachpunkte von Webservices gelten mangelnde Sicherheit durch fehlende Verschlüsselung und Performanzeinschränkungen durch die XML-Verarbeitung [Pap08; Jos08]. Für diese Bereiche bestehen allerdings Abhilfen. So lässt sich Sicherheit sowohl auf Ebene des Transportweges, z. B. durch Verwendung von Secure Hypertext Transfer Protocol (HTTPS), als auch auf Nachrichtenebene durch die Verwendung des WS-Security-Standards [LKN+06], implementieren. Einem Performanzverlust durch erhöhte Nachrichtengröße der XML-Codierung lässt sich durch Verwendung des binären SOAP-Nachrichtenformates Message Transmission Optimization Mechanism (MTOM) und der binären XML-Repräsentation XML-binary Optimized Packaging (XOP) entgegenwirken [GL10].

Bei der Frage nach Fremdbezug oder Eigenerstellung könnte es zum Beispiel für Datenschutz-sensible Domänen die Vorgabe geben, dass bestimmte oder alle unternehmensinternen Daten gar nicht durch externe Stellen verarbeitet werden dürfen. Somit wäre der Fremdbezug für derartige Funktionen ausgeschlossen. Andererseits könnte es die strategische Vorgabe in einem Unternehmen geben, möglichst viel IT auszulagern, falls diese nicht als Kernkompetenz angesehen wird und das Kerngeschäft davon entlastet werden soll [SFF+06]. Neben solchen grundsätzlichen Vorgaben umfassen die Kriterien für die Entscheidung über den Fremdbezug als Service auch technische, ökonomische und organisatorische Aspekte [BHB09]. Zur ökonomischen Fundierung umfangreicher Outsourcing-Fragestellungen kann z. B. die Transaktionskostentheorie herangezogen werden [Arr69; Ouc80]. Hiermit lassen sich die ökonomischen Konsequenzen von Eigenerstellung und Marktbezug verdeutlichen. Auch IT-Outsourcing lässt sich mittels der Transaktionskostentheorie beurteilen [BHB09]. Bei Webservices ist es technisch aufgrund des standardisierten Interfaces einfach möglich, einen für die Eigennutzung erstellten Service auch extern anzubieten. Daher erweitert sich die klassische Frage nach Eigenerstellung oder Fremdbezug zu einer „make-and-sell or buy"-Entscheidung [HHZ11]. Außerdem kann der ressourcenbasierte Ansatz Hinweise auf die Vorteilhaftigkeit einer Sourcingentscheidung geben [Pet93]. In diesem Modell gelten strategisch wichtige, nicht imitierbare Ressourcen, also auch IT, als entscheidend für den Unternehmenserfolg und dürfen nicht ausgelagert werden.

Schließlich stellt sich die Frage, ob die Funktionen, die für einen Servicebezug geeignet sind, überhaupt extern oder intern als Services angeboten werden.

Für externe Services lässt sich dies durch spezialisierte Suchmaschinen [HLV07] klären, die eigentlich für diesen Zweck vorgesehenen UDDI-Verzeichnisdienste [BDE+02] haben diesen Zweck in der Praxis nicht erfüllt [25]. Unternehmensinterne Services sollten per Dokumentation oder in unternehmensinternen UDDI-Verzeichnissen [DJMZ05] auffindbar sein.

Anforderungen an benötigte Funktionen

Um die Anforderungen an die benötigten Funktionen näher zu spezifizieren, können im Fall der Nutzung von Webservices die vom Standardisierungsgremium Organization for the Advancement of Structured Information Standards (OASIS) aufgestellten Kriterien herangezogen werden [BEL+12]. Diese umfassen die Themenfelder „Service Interface", „Service Reachability", „Service Functionality" und „Policies and Contracts, Metrics, and Compliance Records". Die Beschreibung des *Service Interface* umfasst die Spezifizierung der Nachrichten, die der Webservice empfangen und als Antworten versenden kann. Sowohl Syntax als auch Semantik der Nachrichten werden unter diesem Punkt festgelegt. *Service Reachability* beschreibt, unter welcher Adresse ein Dienst erreicht werden kann und mit welchem Protokoll die Nachrichten ausgetauscht werden. Nach diesen beiden technischen Punkten beschreibt *Service Functionality*, was der Service inhaltlich bewirkt und welche Auswirkungen Aufrufe haben. Der Punkt *Policies and Contracts, Metrics, and Compliance Records* beschreibt Zusicherungen über Eigenschaften des Services. Hierzu zählen typischerweise Service Level Agreements (SLAs). Je nach Bedeutung des Dienstes für die zu erstellende Software kann eine detailliertere Analyse der Servicequalität, wie bei der Entwicklung als WOA, nötig sein, die funktionale und nicht-funktionale Anforderungen unterscheidet und jeweils präzisiert [Thi11]. Die nicht-funktionalen Anforderungen betreffen Verfügbarkeit, Performanz, Belastbarkeit/Nutzung, Sicherheit, Preis und Kosten sowie Platzierung. Die funktionalen Anforderungen beschreiben Ein- und Ausgaben, Vor- und Nachbedingungen, Antwortverhalten, Fehlerverhalten und ggf. Vorschriften über zu verwendende Technologien. Gelten hinsichtlich der Technologie externe Vorgaben zur Verwendung von Komponenten, die nicht mit der hier gezeigten XML-basierten Architektur kompatibel sind, so müssen entsprechende Adapter entworfen werden. Soll bspw. ein Präsentationsframework verwendet werden, das kein XML-basiertes Datenmodell verwendet, so muss ein Adapter die XML-Dokumente zur Kommunikation mit dieser Schicht in das zu verwendende Zielformat konvertieren. Dieses Vorgehen widerspricht allerdings der Idee, durchgängig XML-Techniken zu verwenden, und fügt Komponenten hinzu, die keinen Beitrag zur fachlichen Ziel-

Tabelle 4.2: Nicht-funktionale Anforderungen an Services, vgl. [Thi11].

Anforderung	Bedeutung
Verfügbarkeit	Durchschnittliche Verfügbarkeit des Dienstes, erlaubte Dauer von Unterbrechungen, mittlere Betriebsdauer zwischen Ausfällen
Performanz	Antwortzeit des Services, Verarbeitungsdauer
Belastbarkeit/Nutzung	Geplante Nutzungsintensität des Services, Anzahl der Anfragen pro Zeiteinheit und deren Datenvolumen
Sicherheit	Zugriffsrechte auf den Service, Absicherung des Transportmediums und der Daten, Datenschutz
Preis und Kosten	Kosten der Servicenutzung, wobei die Art der Bepreisung je nach Geschäftsmodell unterschiedlich ist
Platzierung	Wo und von wem der Service betrieben wird

setzung der Anwendung liefern. Daher sollte die Auswahl solcher Komponenten nur als Notlösung betrachtet werden.

Tabelle 4.2 erläutert die nicht-funktionalen Anforderungen und Tabelle 4.3 erläutert die funktionalen Anforderungen.

Erforderliche Ausprägungen der Anforderungen für den jeweils beschriebenen Service werden tabellarisch erfasst, um damit passende Anbieter zu identifizieren oder um für unternehmensintern zu betreibende Services als Grundlage für Erweiterung bestehender oder erstmalige Implementierung zu dienen.

Die vorgestellten Ansätze enthalten die Beschreibung der Datenstrukturen in den Unterpunkten *Service Interfaces* sowie *Ein- und Ausgaben*. Dies erscheint bei einer Fokussierung auf Services angemessen. Da im Rahmen der XML-basierten Architektur dem Datenmodell als Softwareschichten übergreifende Struktur eine besondere Bedeutung zukommt, wird in Abschnitt 4.3.2 gesondert darauf eingegangen.

Falls eine Funktion nicht als selbst- oder fremdbetriebener Service eingebunden, sondern innerhalb der Software implementiert werden soll, tritt an die Stelle des *Service Interface* die Signatur der Funktion. Die unter *Service Reachability* zusammengefassten Fragen der technischen Erreichbarkeit der Funktion reduzieren sich auf die Aufteilung der Softwarefunktionen in Bibliotheken. Die Beschreibung der *Service Functionality* bleibt hingegen vollständig relevant und

Tabelle 4.3: Funktionale Anforderungen an Services, vgl. [Thi11].

Anforderung	Bedeutung
Ein- und Ausgaben	Ein- und Ausgabedaten inkl. Format und Transportprotokoll
Vor- und Nachbedingungen	Voraussetzungen für die Nutzung des Services und Angaben über die Auswirkungen eines Serviceaufrufs
Antwortverhalten	Synchrones oder asynchrones Antwortverhalten des Servers
Fehlerverhalten	Kategorien von Fehlern, die auftreten können, und der Umgang mit diesen
Technologieverwendung	Falls eine bestimmte Technologie verwendet werden muss, wird diese Vorgabe hier festgehalten

muss noch erweitert werden. Bei Benutzung von Webservices beschränkt sie sich auf die Beschreibung der gewünschten Funktionen und deren „Real World Effects“ [BEL+12]. Neben dieser externen Sicht auf den Service muss bei Eigenrealisierung auch die interne Funktionsweise entworfen werden. Der Aspekt *Policies and Contracts, Metrics, and Compliance Records* umfasst vertragliche Regelungen zur Bereitstellung des Services. Für die Eigenrealisierung sind die Punkte insofern entbehrlich, als dass der Nutzer der Funktionen gleichzeitig den Betrieb verantwortet. Trotzdem erscheint es sinnvoll, Kernaspekte wie benötigte Performanz und Verfügbarkeit festzuhalten, um die Funktionen bei der Eigenrealisierung entsprechend zu implementieren.

Definition der anzubietenden Schnittstellen

Neben den von der Software zu nutzenden Funktionen werden die als Service anzubietenden Funktionen festgelegt. Hierzu können wieder die Kriterien der OASIS [BEL+12] oder, falls eine präzisere Darstellung benötigt wird, die aus dem WOA-Konzept [Thi11] bekannten Anforderungen herangezogen werden. Für die Spezifizierung der anzubietenden Schnittstellen sind diese Kriterien nicht nur aus Sicht des Servicenutzers zu interpretieren, sondern auch aus Anbietersicht. Als Kriterium für ein sinnvolles Aufteilen von Funktionen auf Services gilt es hierbei, geringe Koppelung der Services und hohe Kohäsion anzustreben [PY02]: Lose Koppelung erfordert, dass ein Service möglichst wenig Abhängig-

keiten zu anderen Services aufweist und eine in sich geschlossene Funktionalität bietet. Geringe Koppelung vermeidet Redundanz, erleichtert Änderungen an einzelnen Services und führt zu einer gesteigerten Wiederverwendbarkeit. Eine hohe Kohäsion bedeutet, dass die Funktionen, die ein Service anbietet, in einem engen sachlogischen Zusammenhang stehen. Hohe Kohäsion erleichtert das Verstehen und die Erweiterung der Services und begünstigt somit ebenfalls Wartung und Wiederverwendbarkeit.

Für die in dieser Arbeit behandelte XML-basierte Architektur ist das Anbieten von Services mittels Web-Schnittstellen naheliegend. Je nach Art der Funktion sind dazu REST- oder SOAP-basierte Services in Betracht zu ziehen [MNS05]. Wie bereits in Abschnitt 2.2.4 dargelegt, kann keine allgemeine Dominanz von SOAP oder REST festgestellt werden.

Im Rahmen der hier vorgestellten XML-basierten Architektur wird grundsätzlich auf REST-Schnittstellen und Ressourcen im XML-Format zurückgegriffen, da diese weniger Implementierungsaufwand für Infrastruktur erfordern. Bei Bedarf sind SOAP-Schnittstellen allerdings ebenfalls implementierbar. Falls andere, möglicherweise weniger standardisierte Schnittstellen, z. B. eine Web API mit weniger strenger Beachtung der Semantik der HTTP-Befehle, erforderlich sind, muss im Einzelfall geprüft werden, wie diese konsistent bereitgestellt werden können.

Der funktionale Entwurf für eine per REST oder Web API erreichbare Funktion ist in Tabelle 4.4 abgebildet. Die Funktion realisiert das Filterschema `Speichere Daten (Subjekt)` des XML-Netzes aus Abbildung 4.5.

User Interface

Neben der Bereitstellung von Services für die Kommunikation mit Maschinen wird auch die Schnittstelle für menschliche Nutzer entworfen. Die Benutzerfreundlichkeit der Mensch-Computer-Interaktion wird in der Literatur bereits seit den 1960er Jahren unter Berücksichtigung des jeweils technisch möglichen Standes untersucht. [LC62] geht von einem textuellen Interface aus, [SG71] beschreibt ein System, das Ausgaben per Sprachsynthese und Eingaben über das Wählen von Ziffern am Telefon realisiert. [Shn82] beschreibt interaktive Systeme mit per Zeigegerät manipulierbaren Elementen. [CO95] untersucht die Rolle der Spracheingabe in der Benutzerschnittstelle. [PSH97] diskutiert die Möglichkeiten der Gestensteuerung.

Grundlegende Prinzipien der Interaktionsgestaltung gelten unabhängig von der zur Realisierung verwendeten Technik und sind bspw. in [Sta07] beschrieben. [Gal02] beschreibt umfangreich die Entwicklung grafischer Benutzerschnitt-

Tabelle 4.4: Funktionale Beschreibung des Services zum Speichern von Formulardaten.

Anforderung	Wert

Eingaben	FormData im ODM-Format,
	REST: per HTTP PUT an die Ressource
	`/$studyoid/$metadataversion/clinicaldata`
	`/$subjectkey/$studyeventdata`
	`/$studyeventrepeatkey/$formdata`
	`/$formrepeatkey` oder
	Web API: per HTTP POST und Parametern
	`studyoid`, `metadataversion`, `subjectkey`,
	`studyevent`, `studyeventrepeatkey`, `form`,
	`formrepeatkey` an die URL
	`/patient/setClinicalData.xql`

Ausgaben	HTTP-Statuscode 200, alternativ Fehlermeldung
Vorbedingungen	Nutzer muss authentifiziert sein
Nachbedingungen	Serviceaufruf speichert Formulardaten des Nutzers
Antwortverhalten	synchron
Fehlerverhalten	User unberechtigt: HTTP-Statuscode 403 Studienteilnehmer oder Ressource unbekannt: HTTP-Statuscode 404
Technologieverwendung	keine Vorgabe

stellen. [Joh07] führt Beispiele korrekter und inkorrekter Anwendung von Graphical User Interface (GUI)-Elementen in User Interfaces auf.

Für den Bereich des Webs, fokussiert auf Webseiten und deren spezifische browserbasierte GUI-Elemente, finden sich Hinweise zur Gestaltung in [Kru06].

Für das Design von Webanwendungen empfiehlt [9] die Orientierung an Aufgaben: Jede Webanwendung soll demnach nur einer Hauptaufgabe dienen. Aufgaben, die damit nicht zusammenhängen, sollen in separate Anwendungen ausgegliedert werden, sodass eine Webseite auch mehrere Anwendungen umfasst. Das dieser Arbeit zugrunde liegende Verständnis einer Webanwendung ist allerdings weiter gefasst, sodass das gesamte, von einer URL erreichbare System als Webanwendung verstanden wird. Die Orientierung an Aufgaben ist jedoch weiterhin ein wichtiger Grundsatz der Benutzerführung. Zusammen mit der Konzentration auf die relevanten Aspekte ergeben sich die folgenden Richtlinien zur Gestaltung [9]:

- Überflüssigen für die Aufgabe nicht benötigten Inhalt minimieren
- Anzahl der Seiten minimieren
- Navigation am Anwendungsfall ausrichten
- Primäre Komponenten sollen stets erreichbar sein
- Konsistentes Design und Nutzung

Weitere Designrichtlinien für Webanwendungen werden in [Hoe11] beschrieben. Da der Fokus dieser Arbeit nicht auf generischem Webdesign liegt, wird im Folgenden nur auf den Entwurf der mit XForms zu realisierenden Elemente eingegangen und es sei für vertiefende Ausführungen dazu auf die genannten und weitere, leicht auffindbare Arbeiten zu diesem Thema verwiesen.

Für den Entwurf von Formularen wird wie in Abbildung 4.7 gezeigt vorgegangen: XForms wird grundsätzlich für diejenigen Bestandteile der Webanwendung verwendet, die Formulare enthalten.

Ausnahmen sind möglich, falls (teilweise) kein XForms verwendet werden soll. Ein möglicher Grund dafür wäre die Anforderung, dass bereits vorhandene HTML-Formulare weiterverwendet werden sollen. Allerdings widerspricht es dem Gedanken der XML-basierten Architektur, wenn die Weboberfläche nicht mit einem XML-Datenmodell entworfen wird, da dann ein Bruch entsteht.

Die Formulare können sowohl für ein statisches als auch für dynamisches Datenmodell entworfen werden. Bei einem statischen Datenmodell ist die Datenstruktur fest, sodass Formularlayout und Verhalten direkt spezifiziert werden können. Bei Vorhandensein eines dynamischen Datenmodells ist a priori nur das XML Schema bekannt und die Erzeugung der Formulare erfolgt durch Anwendung von Transformationsregeln, was sich als Metaprogrammierung auffassen lässt [TLW03]. Dafür müssen Regeln entworfen werden, die angeben, wie Layout und Verhalten für die zu erwartenden Datenelemente erzeugt werden. Diese können tabellarisch und grafisch notiert werden. Tabelle 4.5 gibt dies für ein statisches Benutzerverwaltungsformular wieder, Abbildung 4.8 zeigt einen entsprechenden grafischen Entwurf. Der Entwurf für ein dynamisches Datenmodell wird in Abschnitt 5.2.4 am Beispiel von klinischen Dokumentationsformularen gezeigt.

Bei der GUI-Entwicklung zeigt sich ein Vorteil des durchgängigen XML-Datenmodells. Für die Präsentationsschicht kann auf das bereits aus den darunterliegenden Schichten bekannte Datenmodell zurückgegriffen werden. Die Interaktionselemente werden unabhängig von den Elementen des Datenmodells ent-

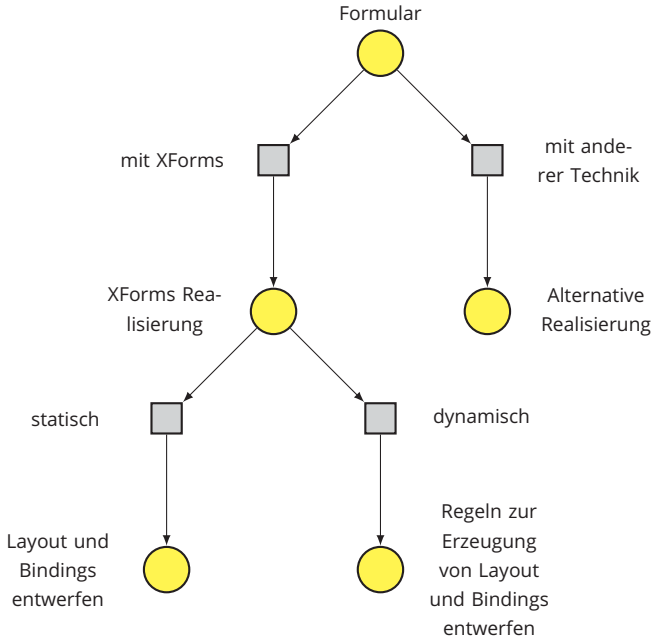


Abbildung 4.7: Entscheidungsprozess für den Entwurf von Formularen. Je nach Voraussetzung werden sie entweder direkt in XForms entworfen oder Regeln zu ihrer Generierung werden festgelegt oder es wird eine alternative Technik verwendet.

Tabelle 4.5: Entwurf eines Formulars zur Benutzerverwaltung.

Datenelement	Bindung	Typ
OID	`odm:User/@OID`	Texteingabe
Type	`odm:User/Type`	Dropdown-Liste
First Name	`odm:User/FirstName`	Texteingabe
Last Name	`odm:User/LastName`	Texteingabe
E-Mail	`odm:User/EMail`	Texteingabe
	`submission:save`	Button
	`reset`	Button

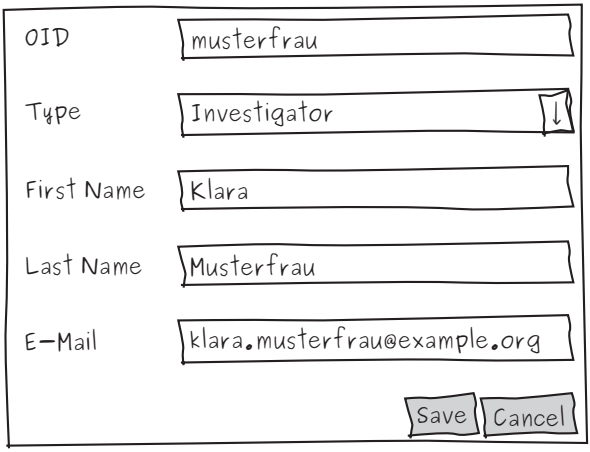


Abbildung 4.8: Grafischer Entwurf für ein Formular zur Benutzerverwaltung.

worfen und über die Bindungselemente mit diesen verknüpft. Durch die Dokumentenorientierung liegen Datenelemente, die häufig zusammen präsentiert werden, oft auch im Datenmodell nah beieinander, was die Handhabung vereinfacht.

4.3.2 Datenmodell

Während der Analysephase wurden bereits die von den Prozessen benötigten Datenobjekte als Klassendiagramme modelliert. Damit wurden die Anforderungen aus fachlicher Sicht festgehalten. Der Entwurf des XML-Datenmodells baut darauf auf.

Im Gegensatz zum relationalen Datenmodell, bei dem Datenbankschemata meist Eigenentwicklungen sind, existiert für viele Domänen aufgrund der Datenaustauschfunktion von XML bereits ein XML Schema. Falls ein geeignetes Schema vorhanden ist, sollte dieses ggf. in erweiterter Form verwendet werden. Einen Ansatzpunkt zum Auffinden eines Schemas bieten Webseiten der in der jeweiligen Domäne aktiven Standardisierungsorganisationen oder XML Schema-orientierte Webseiten wie *XML Standards Library* [23]. Eine Übersicht von XML Schemata diverser Domänen ist auch in [BD13] zu finden.

Wenn ein Schema für einen Anwendungsfall existiert, aber im Detail von den Anforderungen der Analyse abweicht, ergeben sich folgende Optionen:

- Erweiterung des Schemas in einem eigenen Namensraum
- Aufnehmen der Anforderungen in den Standard
- Rückkehr zu Analysephase und Modifikation
- Eigenentwicklung

Die Erweiterung des Schemas wird in vielen Fällen die praktikabelste Lösung sein, um ein Datenmodell zu erhalten, das sowohl dem Standard entspricht als auch den eigenen Anforderungen genügt. XML Schemata einiger Domänen, wie z. B. ODM für klinische Studien oder DocBook [Wal09] für technische Dokumentationen, eignen sich gut für Erweiterungen, da sie im Hinblick darauf konzipiert wurden. Generell lassen sich XML Schemata, die globale Elementdefinitionen, globale komplexe Typen und Gruppen verwenden, gut erweitern [Vli03]. Namensräume in XML Schema bieten eine wichtige Unterstützung für Erweiterungen. So sorgt ein eigener Namensraum der Erweiterungen für eine saubere Trennung standardkonformer von eigenen Elementen. Um ein standardkonformes Dokument zu erhalten, lassen sich alle eigenen Elemente einfach über den Namensraum identifizieren und entfernen. Sollte die Einführung eines eigenen Namensraums unerwünscht sein, so existieren auch Strategien zur Erweiterung von Schemata in einem einzigen Namensraum [Wal10].

Falls die Erweiterungen für viele Nutzer des Standards relevant sein könnten, empfiehlt es sich, diese in den offiziellen Standardisierungsprozess einzubringen. Allerdings handelt es sich dabei um eine langfristige, zusätzliche Maßnahme, da die meisten Standards selten aktualisiert werden, für ODM existieren im Jahr 2013 bspw. erst sechs verabschiedete Versionen seit der ersten Veröffentlichung im Jahr 2000.

Falls die analysierten Anforderungen nicht durch bestehende Schemata oder Erweiterungen dieser realisierbar sind, sollte überlegt werden, warum andere Anwender ohne diese Eigenschaften des Datenmodells auskommen. Ggf. kann bei einer Rückkehr in die Analysephase eine alternative Modellierung der Anforderungen vorgenommen werden, so dass sie zu existierenden Schemata kompatibel sind. Hierbei ist abzuwägen, ob eine Orientierung am standardisierten Schema oder die Verwendung eines selbst entwickelten Schemas vorteilhafter ist.

Falls die Verwendung eines bestehenden, standardisierten Schemas mit oder ohne Anpassungen nicht möglich ist, muss eine Eigenentwicklung vorgenommen werden. Ein möglicher Grund dafür, dass sich ein standardisiertes XML

Schema nicht gut erweitern lässt, ist, dass das Schema nach dem „Matrjoschka-Design“ [SWW11a] entwickelt wurde, bei dem komplexe Elemente mittels verschachtelter komplexer Typen definiert werden und daher nicht wiederverwendbar sind. Für eine Eigenentwicklung wird das Klassendiagramm der Analysephase weiter detailliert und zu einem Schemaentwurf ausgebaut.

Entwurf von XML Schema Obwohl viele XML-DBMS auch schemalose XML-Dokumente speichern können, ist der Entwurf der Datenstrukturen als XML Schema sinnvoll, da nur bei Vorliegen eines Schemas Konsistenzprüfungen vorgenommen werden können. Zu XML Schema bietet die Literatur umfangreiche Beschreibungen [Vli03; SWW11a]. Diese stellen die verfügbaren Elemente der Sprache XML Schema und ihre Verwendung vor, aber diskutieren die Frage guter Schemaentwürfe nicht ausführlich. In Anlehnung an die aus dem relationalen Entwurf bekannten Normalformen, stellt [AL04] einen Normalisierungsalgorithmus für XML-Dokumente vor, der in [YJ08] erweitert wird. Für die Verwendung zum Entwurf neuer XML Schema sind diese Ansätze aus praktischer Sicht problematisch: Sie basieren auf der Analyse bestehender Daten und die dabei verwendeten Tools sind nicht öffentlich verfügbar. Die Idee der Algorithmen ist die Erkennung funktionaler Abhängigkeiten im Schema und die Normalisierung des Schemas zur Reduzierung der funktionalen Abhängigkeiten. Wie bei der Normalisierung relationaler Schemata soll auch hierbei eine Reduzierung der Redundanz und somit Robustheit gegenüber Einfüge-, Lösch- und Änderungs-Anomalien erreicht werden.

In Abwesenheit anwendbarer XML-orientierter Regeln zur Normalisierung wird im Folgenden gezeigt, wie die relationalen Normalisierungsregeln auf XML-Dokumente angewendet werden können. Das Ziel ist, XML Schemata so zu entwickeln, dass sie normalisiert vorliegen. Die Normalisierung zur BCNF stellt ein gutes Design dar, weil keine Update-Anomalien auftreten und Redundanzfreiheit sichergestellt ist [AL05]. Andererseits garantieren nur die Transformationen bis zur 3NF, dass funktionale Abhängigkeiten während der Normalisierung erhalten bleiben. Die 3NF garantiert von den abhängigkeitserhaltenden Normalformen die geringste Redundanz [KL06]. Des Weiteren sind in der Praxis die meisten Relationenschemata, die in der 3NF sind, auch in der BCNF [EN02]. Daher wird im Rahmen dieser Arbeit ein Äquivalent zur 3NF für den Entwurf von XML Schemata angestrebt.

Bei der Normalisierung spielen Schlüsselattribute des Relationenschemas eine zentrale Rolle. In XML-Dokumenten sind Schlüsselattribute im Gegensatz zu relationalen Schemata keine natürlichen Eigenschaften des Paradigmas, da sich

```
1  <!-- kein atomarer Wertebereich -->
2  <User OID="mustermann" UserType="Investigator">
3    <FirstName>Max</FirstName>
4    <LastName>Mustermann</LastName>
5    <Email>max@example.org, mustermann@example.org</Email>
6  </User>
7  <!-- Atomare Wertebereiche -->
8  <User OID="mustermann" UserType="Investigator">
9    <FirstName>Max</FirstName>
10   <LastName>Mustermann</LastName>
11   <Email>max@example.org</Email>
12   <Email>mustermann@example.org</Email>
13 </User>
```

Programm 4.1: Dokument, bei dem eine Verletzung der 1NF vorliegt und Transformation in ein Dokument, das diese einhält.

die Identität eines Elementes im XML-Dokument auch bei Struktur- und Wertgleichheit seines Inhaltes mit einem weiteren Element von diesem unterscheidet. Oft werden aber künstliche Schlüsselattribute verwendet, um Elemente zu sinnvoll identifizieren zu können. Im Folgenden wird die Normalisierung anhand einer Dokumentinstanz dargestellt, da das Prinzip damit leichter als mit den dazugehörenden XML Schema-Dateien nachzuvollziehen ist.

Die 1NF fordert einen atomaren Wertebereich für jedes Attribut und gilt als Teil der Definition einer Relation [EN02]. Übertragen auf XML Schema bedeutet dies, dass die Inhalte von Attributen und Elementen ebenfalls atomar sein müssen. Programm 4.1 zeigt ein Dokument, bei dem die 1NF verletzt ist und die Umwandlung in ein Dokument, das die 1NF einhält. Das Element `Email` in Zeile 5, das zwei E-Mail-Adressen enthält, wird in Zeile 11 und 12 in zwei Elemente mit atomaren Werten transformiert.

Die 2NF erfordert, dass die 1NF eingehalten ist und kein Nichtschlüsselattribut funktional von einer Teilmenge der Schlüsselkandidaten abhängig ist. Die Analogie zu XML Schema ist, dass ein übergeordnetes Element als Schlüssel verstanden wird, von dem der Inhalt dieses Elements abhängig ist. Angewendet auf XML Schema ergibt sich aus 2NF die Forderung, dass kein Bestandteil des Elementinhaltes nur von einem Teil der das Element identifizierenden Eigenschaften abhängen darf. Programm 4.2 zeigt eine Verletzung dieser Forderung und

```
1  <!-- Kindelemente hängen nur von einem Teil der das
       Elternelement identifizierenden Eigenschaften ab -->
2  <Schedule UserRef="musterfrau" Location="clinA" date="
       2013-01-01-">
3    <Name>Klinik A</Name>
4    <LocationType>Site</LocationType>
5    <Start>09:00:00</Start>
6    <End>17:30:00</End>
7  </Schedule>
8  <!-- Beseitigung der funktionalen Abhängigkeit -->
9  <root>
10   <Schedule UserRef="musterfrau" LocationRef="clinA" date
         ="2013-01-01-">
11     <Start>09:00:00</Start>
12     <End>17:30:00</End>
13   </Schedule>
14   <Location OID="clinA">
15     <Name>Klinik A</Name>
16     <Type>Site</Type>
17   </Location>
18 </root>
```

Programm 4.2: Dokument, bei dem eine Verletzung der 2NF vorliegt und Transformation in ein Dokument, das diese einhält.

die Umwandlung in ein konformes Dokument. Sinn des Elementes `Schedule` ist es, die Einsatzzeit eines Users in einer Klinik an einem bestimmten Datum festzuhalten. Das Element wird also durch den Nutzer, die Klinik und das Datum identifiziert. Die Elemente `Name` und `LocationType` in Zeilen 3 und 4 beziehen sich auf den Einsatzort und sind somit nur vom Attribut `Location`, das nur einen Teil des Schlüssels darstellt, funktional abhängig: `Location` → `Name,LocationType`. Wenn das Schema wie im Beispiel vorsieht, sie im Element `Schedule` zu speichern, muss dies für jedes dieser Elemente geschehen und ist somit redundant. Um die 2NF zu erfüllen, wird ein neues Element `Location` eingeführt, das die davon abhängigen Eigenschaften enthält und referenziert wird.

Die 3NF erfordert zusätzlich zu den Bedingungen der 2NF, dass kein Nichtschlüsselattribut von einem Schlüsselkandidaten transitiv abhängig sein darf.

Übertragen auf XML Schema bedeutet dies, dass ein Element nur von den es einschließenden Element abhängen darf und nicht von seinen Geschwisterelementen. Programm 4.3 zeigt eine Verletzung der 3NF und deren Behebung. Es zeigt einen fiktiven Stammdatensatz für einen Patienten. Es sollen der Name und die Unterbringung des Patienten erfasst werden. Für die Erfassung der Unterbringung dienen die Elemente `Location`, `Street` und `City` in den Zeilen 5 bis 7 Das Element `Location` hängt vom Patienten ab, die Elemente `Street` und `City` hängen von `Location` ab, und somit nur transitiv vom Patienten: `PatientOID` → `FirstName,LastName,Location`; `Location` → `Street,City`. Für einen weiteren Patienten, der in Klinik A untergebracht ist, müssten sie wiederholt werden, wodurch Redundanz entsteht. Um die 3NF zu erreichen, wird die Klinik in ein neues Element `Location` in Zeile 15 ausgelagert und vom Element `Patient` in Zeile 11 referenziert.

Um Redundanzen im Schemaentwurf zu vermeiden, scheint eine Orientierung an den gezeigten, auf XML übertragenen relationalen Konzepten sinnvoll.Allerdings kann es Fälle geben, in denen es möglich ist, von einer Normalisierung abzuweichen und Redundanzen zuzulassen, um bspw. Geschwindigkeitsvorteile zu erzielen.

Beispiel Für die eigentliche Studiendokumentation existiert das bereits in Abschnitt 3.4.4 präsentierte ODM Schema. Notwendige Erweiterungen dazu werden in einem eigenen Namensraum vorgenommen, was in Abschnitt 5.2 im Rahmen der Entwicklung der Software x4T gezeigt wird.

Für das in Abbildung 4.3 analysierte Datenmodell zur Anbindung eines EDC an ein KIS existiert kein standardisiertes Schema. Daher wird in Abbildung 4.9 ein entsprechender Entwurf gezeigt. Die Elemente Zugriffsberechtigung und Studie werden im Element des Studienteilnehmers abgebildet. Da die Elemente User und Patient bereits im KIS vorhanden sind, benötigen sie keinen Entwurf. Zeitplan und Status sind als Teile des zugehörigen Dokumentationselements entworfen. Die mit dem Präfix xs gekennzeichneten Datentypen sind durch XML Schema definiert, der Datentyp oid wird so wie der gleichlautende Datentyp aus ODM definiert und docstatus wird als Enumeration im Diagramm definiert.

Die Umsetzung des Entwurfsdiagramms in XML Schema steht am Übergang von Entwurfs- und Implementierungsphase. Einerseits kann das Erstellen der Schemata als Teil der Implementierung aufgefasst werden, da dabei zu benutzender Programmcode entsteht. Auf der anderen Seite steht der Entwicklungsprozess nach der Umsetzung eines eigenen Schemas auf demselben Stand wie nach der alternativen, zur Entwurfsphase gehörenden Auswahl eines vorhan-

```
<!-- Elemente Street und City hängen nur über das
     Attribut Location vom Schlüsselattribut ab -->
<Patient PatientOID="doe">
  <FirstName>John</FirstName>
  <LastName>Doe</LastName>
  <Location>Klinik A</Location>
  <Street>Albert-Schweitzer-Campus 3</Street>
  <City>Münster</City>
</Patient>
<!-- Beseitungung der transitiven Abhängigkeit durch
     Auslagern in das neue Element Location und Anlegen
     einer Referenz-->
<root>
  <Patient PatientOID="doe" LocationRef="clinA">
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
  </Patient>
  <Location OID="clinA">
    <Name>Klinik A</Name>
    <Street>Albert-Schweitzer-Campus 3</Street>
    <City>Münster</City>
  </Location>
</root>
```

Programm 4.3: Dokument, bei dem eine Verletzung der 3NF vorliegt und Transformation in ein Dokument, das diese einhält.

denen standardisierten XML Schemas. Ohne praktische Auswirkungen auf die Entwicklung wird die Erstellung der XML Schema hier als Abschluss der Entwurfsphase betrachtet, da somit unabhängig von deren Herkunft alle benötigten XML Schema-Dateien beim Übergang zur Implementierungsphase vorliegen. Die Verwendung dieser Schemadateien in einem XML-DBMS ist dann eindeutig der Implementierungsphase zuzurechnen. Das aus dem Entwurf in Abbildung 4.9 abgeleitete XML Schema und ein dazu passendes Dokument finden sich in Abschnitt A.1.

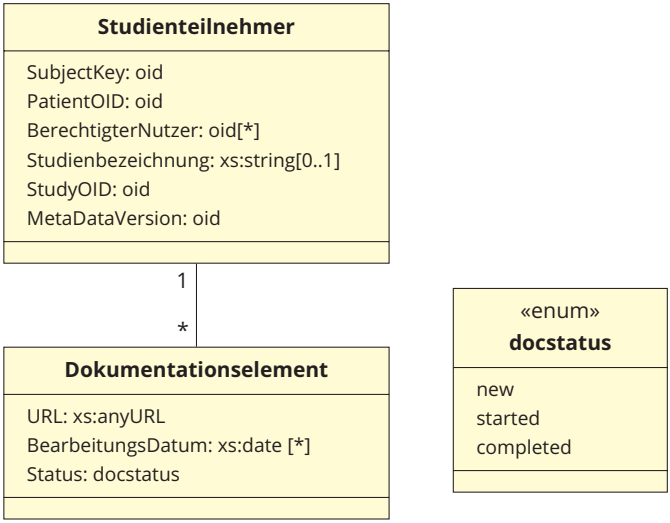


Abbildung 4.9: Entwurf der Datenobjekte in UML als Verfeinerung und Umstrukturierung der Analyseergebnisse.

4.3.3 Organisation

Der Entwurf der organisatorischen Aspekte basiert auf der analysierten Organisationsstruktur. Die Organisationsstruktur wird soweit verfeinert, dass sie die Implementierung von funktionsübergreifenden Aspekten ermöglicht. Dazu gehört das Rechtemanagement.

Aus technischer Sicht ist ein rollenbasiertes Rechtemanagement [FK92] wenig flexibel und führt schnell zu einer Vielzahl an Rollen [KCW10]. Für jede Kombination von Eigenschaften, die zur Rechtevergabe relevant ist, muss eine eigene Rolle erzeugt werden.

In einem standort- und mandantenfähigen System zur Datenerfassung klinischer Studien müsste es z. B. für jede analysierte Rolle eigene technische Rollen geben, s. Tabelle 4.6. Für m Studien mit je n Standorten ergibt dies $1 + m + 3mn$ benötigte Rollen. Bei Veränderungen des Rechtemodells müssen also potenziell viele Rollen modifiziert werden, was die Wartbarkeit beeinträchtigt.

Ein vielfach passenderer Ansatz ist die Verwendung eines attributbasierten Rechtemanagements [PDMP05]. Dabei werden den Benutzern Attribute zugewiesen und der Zugriff auf Ressourcen wird anhand von Regeln entschieden.

Tabelle 4.6: Benötigte technische Rollen bei Implementierung eines rollenbasierten Zugriffmodells.

Fachliche Rolle	pro Studie	pro Standort
Systemadministrator		
Studienadministrator	+	
Standortadministrator	+	+
Standortauswerter	+	+
Datensammler	+	+

Wenn ein Benutzer das Attribut Systemadministrator aufweist, so werden ihm alle Rechte eingeräumt. Die anderen Rollen ergeben nur in Kombination mit weiteren Attributen, nämlich für Studie und Standort, Sinn. Weist ein Benutzer die Kombination aus Studienadministrator und der entsprechenden Studie auf, so werden erhält er alle Rechte zur Verwaltung dieser einen Studie. Für den Standortadministrator muss noch das passende Standortattribut vorhanden sein. Benutzer, die als Standortauswerter oder Datensammler arbeiten, weisen ebenfalls diese beiden Attribute auf, erhalten aber laut Regelwerk weniger Rechte. Die Regeln werden zentral festgelegt, so dass eine Änderung der Rechte leicht möglich ist. Neben der Nachbildung klassischer Rollenkonzepte lassen sich mittels attributbasierter Zugriffskontrolle beliebige Eigenschaften des Nutzers oder der Umgebung zur Rechteprüfung heranziehen. So könnte eine Regel festlegen, dass Zugriffe nur in einem bestimmten Zeitfenster erlaubt sind.

Aufgrund dieser Flexibilität wird im Rahmen der hier beschriebenen XML-basierten Architektur ein attributbasiertes Rechtemanagement verwendet, das sich an das Muster „Metadata-based Access Control“ [PFMP04] anlehnt. Es wird eine Modifikation daran vorgenommen, mittels derer Zugriffsregeln zusammengefasst werden können, um ein derartiges Bündel an Regeln leichter wiederverwenden zu können. Dazu werden folgende Erweiterungen eingeführt:

- *Resource:* Unter *Resource* werden sämtliche zugreifbare Einheiten verstanden, die durch das Rechtemanagement geschützt werden.
- *Policy:* Eine *Policy* enthält der Resource zugeordnete Zugriffsregeln. Diese sind mittels logischem ODER verknüpft, d. h. bei Zutreffen mindestens einer Regel wird der Zugriff gewährt.
- *Clause:* Das Element *Clause* bezeichnet eine Zugriffsregel. Diese enthält

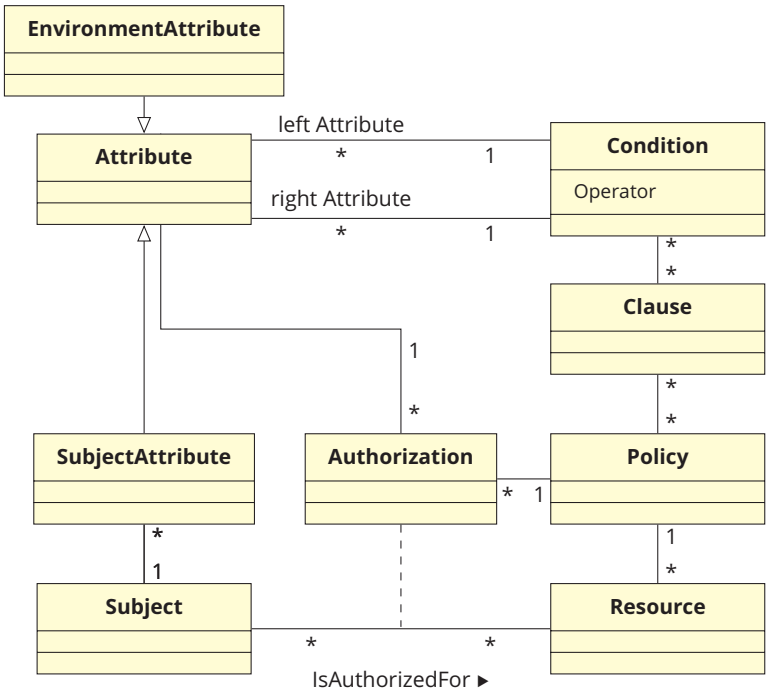


Abbildung 4.10: Struktur des Rechtemanagements der hier vorgestellten XML-basierten Architektur.

mittels logischem UND verknüpfte Bedingungen, d. h. nur wenn alle Bedingungen zutreffen, wird die *Clause* als zutreffend gewertet.

- *Condition:* Mit *Condition* wird eine Bedingung bezeichnet, die für ein Attribut gelten muss. Dazu werden ein linkes und ein rechtes Attribut sowie ein Vergleichsoperator angegeben.

Die Struktur des gesamten Rechtemanagements ist in Abbildung 4.10 wiedergegeben. Zentral ist die Assoziationsklasse *Authorization* zwischen dem *Subject* genannten Benutzer und *Resource*. Durch Auswertung der der *Resource* zugeordneten *Policy* mittels dem *Subject* zugeordneter oder sonstiger Attribute, wird eine Entscheidung über das Zugriffsrecht gefällt.

Zum Entwurf der Berechtigungen werden für den Zugriff auf Ressourcen re-

Tabelle 4.7: Entwurf der Rechte.

Resource

Resource	Policy
Studie hinzufügen	add-study
Standort hinzufügen	add-location

Policy

Policy	Clause
add-study	user-is-admin
add-location	user-is-admin
add-location	user-is-pi

Clause

Clause	Condition
user-is-admin	subject-is-admin
user-is-pi	subject-is-pi
user-is-pi	subject-role-location-equals-resource

Condition

Condition	leftAttribute	Operator	rightAttribute
subject-is-admin	subject.role	=	'admin'
subject-is-pi	subject.role	=	'pi'
subject-role-location-equals-resource	subject.role.studyOID	=	resource.studyOID

levante Attribute identifiziert und tabellarisch notiert, Tabelle 4.7 zeigt ein Beispiel. Fachliche Rollen werden dabei aus Attributkombinationen gebildet. Hat ein Nutzer bspw. das Attribut Studienadministrator, so gehört dazu auch das Attribut, das eine Referenz auf die entsprechende Studie enthält. Diese Attributkombinationen werden als *Clause* zusammengefasst.

Die Überprüfung der Rechte findet in Webanwendung vor dem Aufruf der jeweiligen Funktionen statt. Gegenüber der Nutzung DBMS-interner Zugriffskontrollen hat dies den Vorteil, dass eine feinere Steuerung möglich ist. So hat ein Nutzer in der Rolle des Datensammlers zwar beim Aufruf eines Formulars Lese- und Schreibrechte für die Daten seines Standortes, darf diese aber nicht über die Exportfunktion herunterladen. Der Controller der Webanwendung, den alle Anfragen durchlaufen, dient als zentrale Stelle zur Rechteüberprüfung. Somit kann an einer Stelle der Anwendung sichergestellt werden, dass nur berechtigte Anfragen aufrufbar und somit ausführbar sind und unberechtigte Anfragen abgelehnt werden und zu einer entsprechenden Fehlermeldung führen.

4.4 Implementierung

Bisher wurden Modelle des zu entwickelnden Softwaresystems erstellt. Die aus der Entwurfsphase hervorgegangenen Modelle dienen als Grundlage für die in der Implementierungsphase erfolgende Umsetzung als Software. Dazu werden zunächst die zu verwendenden Standard-Softwarekomponenten ausgewählt. Für Software nach der hier vorgestellten XML-basierten Architektur sind dies mindestens ein Web-fähiges XML-DBMS mit XQuery-Interpreter und ein XForms-Interpreter sein. Folgend wird eine Entwicklungsinfrastruktur bestimmt und aufgesetzt. Dazu gehören zentrale Komponenten wie ein Code-Repository und ein Integrationsmechanismus. Je nach Vorgehensmodell werden dann die Testfälle oder die Software programmiert. Dabei werden die Entwürfe für die einzelnen Schichten umgesetzt und auf unterschiedlichen Integrationsstufen werden Tests durchgeführt.

4.4.1 Auswahl der Komponenten

Die meisten derzeit verfügbaren Systemkomponenten und darauf aufsetzende Frameworks folgen dem relationalen Paradigma für Realisierung der Persistenzschicht und dem objektorientierten für die Realisierung der Geschäftslogik. Bei Verwendung der hier vorgestellten XML-basierten Architektur ist die Auswahl der Komponenten gegenüber herkömmlicher Entwicklung eingeschränkt. Die Auswahl muss auf Grundlage der zum XML-Entwurf kompatiblen verfügbaren Komponenten erfolgen.

Minimal notwendig für die hier betrachtete XML-basierte Architektur sind ein XML-DBMS mit XQuery-Interpreter und ein XForms-Interpreter. Weitere ggf. benötigte Komponenten werden nach Bedarf ausgewählt. Das Aufstellen von Kriterienkatalogen zur Auswahl von Software ist komplex [CFQ07] und oft nur mittels umfangreicher Vorgehensmodelle vollständig zu bewerkstelligen [FC03]. Die folgenden Ausführungen beschränken sich daher auf die Beschreibung einiger Eigenschaften, die als relevante Unterscheidungskriterien für die verfügbaren XML-DBMS dienen.

XML DBMS Die XQuery-Spezifikation wird durch unterschiedliche Interpreter implementiert, die das W3C in einer Liste[2] dokumentiert. Nicht alle dort geführten Implementierungen sind Teil eines aktuell gepflegten DBMS. Einige

[2] http://www.w3.org/XML/Query/#implementations

hier nicht in Frage kommende Implementierungen stellen reine In-Memory Interpreter oder Bibliotheken dar.

Zunächst sollte die Unterstützung des XQuery-Standards beachtet werden, da dieser zentral für die hier vorgestellte XML-basierte Architektur ist. Die Korrektheit einer Implementierung kann mittels der *XML Query Test Suite (XQTS)* [39] überprüft werden. XQuery-implementierende DBMS unterscheiden sich durch die unterstützte Version und hinsichtlich der Vollständigkeit der Implementierung. XQuery enthält sowohl in Version 1.0 als auch in den Entwürfen zu Version 3.0 einige im Standard als optional deklarierte Bestandteile. Wichtig ist die Unterstützung der *Module Features*, da nur dann modularisierte Programme erstellt und Bibliotheksmodule verwendet werden können, was zur Beherrschung von umfangreichem Programmcode nötig ist. Unterstützung des *Serialization Feature* ist ebenfalls nötig, um eine textliche Ausgabe zu erhalten. Falls Version 3.0 gewählt wird, ist die Unterstützung von *Higher-Order Function Feature* eine wünschenswerte Eigenschaft, um Funktionen höherer Ordnung zu verwenden und somit verstärkt das funktionale Programmiermodell nutzen zu können.

Für das im XQuery-Standard nicht spezifizierte Ändern von Dokumenten muss eine *Update-Sprache* vorhanden sein. Zur Verbesserung der Portabilität empfiehlt sich dazu die Unterstützung mit der vom W3C standardisierten XQuery Update Facility, akzeptabel sind aber auch implementierungsspezifische Sprachen.

Je nach Anwendungszweck kann die Bereitstellung weiterer, nicht standardisierter Funktionen durch das DBMS als Application Server nötig sein bzw. die Implementierung der Webanwendung vereinfachen. Hierzu zählt die Bereitstellung von Diensten wie einem Scheduler, von Schnittstellen zum Dateisystem und zu Kommunikationsdiensten wie E-Mail oder HTTP und die Bindung an andere Programmiersprachen. Die Transaktionsmodelle von XML-DBMS basieren auf unterschiedlichen Konzepten, die jeweils für bestimmte Zugriffsarten optimiert sind [SLJ12]. Bei einigen DBMS kann der Isolationgrad zur Steigerung der Performanz konfiguriert werden, eXist-db verzichtet auf vollständige Isolation, indem Leseoperationen stets erlaubt werden. Eine unvollständige Isolation kann zum Problem des „Dirty Read“ [Vos08] führen, wenn dieses Verhalten bei der Implementierung der Anwendung nicht berücksichtigt wird.

Je nach erwarteter Skalierbarkeit der Applikation sollten die Fähigkeiten des DBMS zur verteilten Ausführung geprüft werden. Die Forschungsergebnisse zur verteilten Ausführung von XML Query [ÖK10; KÖD11] haben derzeit noch keinen Eingang in die Implementierungen gefunden, es stehen aber bei einigen

DBMS eine klassische Master/Slaver-Replikation [Ens78] oder produktspezifische Möglichkeiten zur Verteilung der Datenbanken zur Verfügung.

Ein weiterer wichtiger Punkt ist die Integration des XML-DBMS und der weiteren Komponenten der Architektur. Der für die Web-Schnittstelle benötigte Webserver kann entweder durch das DBMS bereitgestellt werden, oder das DBMS stellt ein Plug-in zur Integration in einen Webserver bereit. Einige DBMS bieten sowohl eine allein lauffähige Version mit integriertem Webserver als auch ein Modul zum Betrieb in einem Webserver bzw. ein Web Archive zum Betrieb in einem Java Servlet Container an.

Einige relevante XML-DBMS und ihre Merkmale sind in Tabelle 4.8 aufgeführt. Die Implementierungsbeispiele im Rahmen der vorliegenden Arbeit wurden anhand von DBMS eXist-db entwickelt, für das eine grundlegende Beschreibung des internen Aufbaus in [Mei03] zu finden ist. Das gezeigte Konzept der XML-basierten Architektur ist allerdings nicht an eXist-db gebunden.

XForms Bei der Integration der für die Benutzerinteraktion benötigten XForms-Interpreter gibt es ebenfalls Unterschiede. Zu unterscheiden sind primär *clientseitig* ausgeführte XForms-Interpreter, die im Browser des Anwenders arbeiten, und *serverseitig* ausgeführte Interpreter. Bei clientseitig ausgeführten Interpretern muss beachtet werden, dass die Kontrolle der Ausführung hierbei nicht mehr im Einflussbereich der Webanwendung liegt und Vorkehrungen gegen „Client State Manipulation"-Angriffe [DKK07] zu treffen sind. Bei serverseitig ausgeführten XForms-Interpretern liegt die Ausführung des Formulars unter der Kontrolle der Webanwendung, indem der Zustand des Formulars verbindlich serverseitig vorgehalten wird und somit clientseitig nur erlaubte Modifikationen stattfinden können. Für die hier vorgestellte XML-basierte Architektur werden serverseitig ausgeführte XForms-Interpreter angenommen. Deren Implementierung kann, wie bei betterFORM, eng mit einem zugrunde liegenden XML-DBMS integriert sein oder als Web Archive, wie bei *Orbeon*[3], unabhängig von der Implementierung der darunterliegenden Schichten arbeiten. Der Grad der Konformität der XForms-Interpreter mit der Spezifikation kann durch Ausführen der der *XForms Test Suite* überprüft werden, oftmals stellen Anbieter von XForms-Interpretern die Testergebnisse bereit. Einige relevante XForms-Interpreter und ihre Merkmale sind in Tabelle 4.9 aufgeführt.

Softwareökosystem Für jede der verwendeten Softwarekomponenten gilt es außerdem, neben den technischen Eigenschaften noch weitere Aspekte zu un-

[3] http://www.orbeon.com/

Tabelle 4.8: Liste von XML-DBMS Implementierungen.

Bezeichnung	XQuery	XQTS 1.0-Konformität [6]	Update-Sprache	HTTP	Replikation	Transaktionsmodell	Lizenz	Dokumentation
BaseX 7.8 http://basex.org/	3.0	99,9 %	XQuery-Update Facility 1.0	ja	nein	ACID	frei (BSD)	[Bas14]
eXist-db 2.1 http://exist-db.org/	1.0 / 3.0 (eingeschränkt)	99,4 %	XQuery Update Extension	ja	Master/Slave	ACD (siehe Text)	frei (LGPL)	[16]
Sedna 3.5 http://www.sedna.org/	1.0	98,8 %	proprietär	mit Apache HTTP Server	nein	ACID	frei (Apache License)	[21]
DB2 11 pureXML http://www.ibm.com/db2/xml	1.0	unbek.	vollst. Dokument ersetzen	mit IBM WebSphere	umfangreich: IBM InfoSphere	ACID	proprietär	[IBM13]
MarkLogic 7 http://www.marklogic.com/	1.0	99,9 %	proprietär	ja	umfangreiche Clusterkonfiguration	ACID	proprietär	[24]
Oracle 12c http://www.oracle.com	1.0	unbek.	Query-Update Facility 1.0	ja	umfangreich: Oracle Streams	ACID	proprietär	[Dra13]
Tamino 8.2.2 http://www.softwareag.com	1.0 (eingeschränkt)	unbek.	proprietär	ja	Master/Slave	ACID	proprietär	[31]

Tabelle 4.9: Liste von XForms-Implementierungen.

Bezeichnung	Ausführung	Konformität gemäß Test-Suite	Lizenz	Dok.
betterFORM 5 RC 3	Server	95–98 % je nach Browser [3]	frei (BSD)	[1]
Orbeon Forms 4.4 CE	Server	58 % (Tests unvollständig) [5]	frei (LGPL)	[2]
XSLTForm 1.0 RC	Client	45 % [4]	frei (LGPL)	[7]
IBM Forms 8.0	Server	unbekannt	proprietär	[8]

Tabelle 4.10: Aktivitätsindikatoren untersuchter DBMS im Jahr 2010.

DBMS	Softwarebeiträge	Beiträge auf der Mailingliste
BaseX	954	950
eXist-db	2565	5429
Sedna	349	298

tersuchen, die Einfluss auf die Vorteilhaftigkeit der Verwendung haben. Dazu zählt das Softwareökosystem, also die Software und die darum bemühten Entwickler, Serviceanbieter und Anwender [BB10]. Komponenten mit einem größeren Ökosystem dürften ausgereifter, verbreiteter und langfristig besser verfügbar sein als solche mit einem begrenzteren. Eine formale Analyse des Softwareökosystems [BJB09] dürfte in praxi meist zu aufwändig sein. Allerdings lässt eine Suche im Web meist eine brauchbare Einschätzung zu. Aktuelle Webseiten, umfangreiche Dokumentation, aktive Mailinglisten oder Diskussionsforen und Berichterstattung über die Software deuten auf ein vitales Ökosystem hin.

Für die unter freier Lizenz stehenden XML-DBMS wurde zu Beginn der x4T-Entwicklung die Aktivität von öffentlichem Support und Entwicklung betrachtet. Als Indikatoren wurden aus Support-Mailinglistenarchiven und Versionsverwaltungssystemen die Anzahl der Beiträge für das Jahr 2010 ermittelt, die in Tabelle 4.10 gezeigt sind.

Lizenzierung Weitere Beachtung sollte die Lizenzierung der Softwarekomponenten erfahren. Daraus ergeben sich Möglichkeiten und Bedingungen für die weitere Verwendung der darauf aufbauenden Software. Bei Verwendung von Komponenten unter freier Lizenz gilt meist eine der in [Lau04] besprochenen Lizenzen, bspw. die GNU Lesser General Public License (LGPL). Bei sonstigen Lizenzen gelten vom jeweiligen Hersteller vorgegebene Vereinbarungen, die sehr unterschiedlich ausgestaltet sein können. Allgemeine Hinweise zur rechtlichen

Lage bei der Herstellung und Verwendung von Software finden sich in [KS91].

Serviceanbieter Für die Funktionen, die laut Entwurf zum Bezug per Service vorgesehen sind, sind entsprechende Dienste zu integrieren. Bei unternehmensintern angebotenen Services ist die Auswahl offensichtlich, sie können direkt verwendet werden. Bei externen Services steht vor der Verwendung die Auswahl des oder der Service-Anbieter. Da in der Praxis keine automatisiert durchsuchbaren Verzeichnisdienste verfügbar sind, muss die Suche nach einem geeigneten Anbieter von Hand erfolgen [Thi11]. Dazu gilt die Webseite ProgrammableWeb [30] als der umfangreichste Dienst [HFT12], weitere sind durch Onlinerecherche leicht auffindbar. Gemäß den im Entwurf aufgestellten funktionalen und nicht-funktionalen Anforderungen sind Services zu identifizieren und auszuwählen. Leistungsbeschreibungen und Preisgestaltung unterscheiden sich dabei je nach Anbieter.

Anbieter von Services bieten meist als SLAs standardisierte Leistungen allgemein zugänglich am Markt an, wodurch sich individuelle und aufwändige Vertragsverhandlungen vermeiden lassen. Hierbei reduziert sich die Vertragsverhandlung auf die Entscheidung, ob und welches Leistungsangebot genutzt werden soll. Darüber bieten einige Anbieter auch individuell verhandelbare Leistungen an.

Unabhängig davon, ob interne oder externe Services genutzt werden, müssen die Aufrufe der APIs an die individuell genutzten Dienste angepasst werden. Zur Erzielung einer loseren Koppelung bietet sich dazu die Verwendung des Entwurfsmusters *Adapter* an [BCG+05].

4.4.2 Aufbau der Entwicklungsinfrastruktur

Die zur Entwicklung einer Software benötigte Infrastruktur als Arbeitsgrundlage der Entwickler und Mechanismen zur Koordination hängen von der Art des Entwicklungsprojekts und dem gewählten Vorgehensmodell ab. Unabhängig davon sind jedoch einige Werkzeuge als Standard zu betrachten:

Ein *Versionsverwaltungssystem* enthält den Programmcode und stellt ihn den Entwicklern zur Verfügung. Änderungen daran werden durch das System protokolliert und vorherige Zustände bleiben wiederherstellbar. Unterschiedliche Zweige der Software können parallel gepflegt werden. Das Integrieren konkurrierender Änderungen durch unterschiedliche Entwickler wird durch die Versionsverwaltung unterstützt. Für agile Methoden, bei denen gemeinsamer Codebesitz der Entwickler gilt [BW08], ist die Verwendung eines Versionsverwal-

tungssystems zwingend nötig, allgemein gilt sie als „best practice“ [Spi05]. Beliebte Versionsverwaltungssysteme sind bspw. *Apache Subversion (SVN)*[4] und *Git*[5]. SVN basiert auf einem zentralen Server, der das offizielle Code-Repository verwaltet. Aus diesem beziehen die Entwickler den Programmcode und übermitteln ihre Modifikationen. So entsteht eine lineare Historie an Versionsständen. Für SVN existiert eine breite Toolunterstützung und umfangreiche Erfahrung im Einsatz. Von Git wird ein dezentraler Ansatz verfolgt: Jeder Entwickler unterhält lokal ein eigenes, vollständiges Code-Repository, in dem entwickelt wird. Bei Bedarf, z. B. bei gemeinsamer Arbeit an einer Funktion, können Daten zwischen den einzelnen Repositories übertragen werden. Die abgeschlossenen Änderungen werden an ein als offiziell deklariertes Projekt-Repository übermittelt. Git bietet umfangreiche Funktionen zum Arbeiten mit unterschiedlichen Entwicklungszweigen und dem damit verbundenen Aufteilen und Verschmelzen von Programmcode. Wenn einzelne Funktionen fertig entwickelt sind, wird deren Programmcode in ein als offiziell deklariertes Code-Repository übertragen. Dieser verteilte Ansatz erlaubt es, erst teilweise fertiggestellten Code zu versionieren, parallel an unterschiedlichen Funktionen zu arbeiten und alternative Implementierungen auszuprobieren, ohne unfertigen Code in das offizielle Projektrepository zu übertragen, erfordert aber auch ein Umdenken der Beteiligten gegenüber der zentralen linearen Versionierung.

Ein *Continuous Integration (CI)*-System integriert und testet permanent Änderungen am Quelltext, die an das Versionsverwaltungssystem übergeben wurden [DMG07]. Die Verwendung von CI ermöglicht es, kontinuierlich sicherzustellen, dass das gesamte entwickelte System lauffähig ist. Zudem erleichtert das regelmäßige Integrieren und Testen kleinerer Änderungen das Auffinden von Fehlern, da der fehlerhafte Code häufig schneller eingegrenzt werden kann. Die seltene Integration größerer Artefakte birgt die Gefahr, dass separat entwickelte Komponenten zueinander nicht kompatibel sind. Somit entsteht dabei das Risiko, dass nach dem geplanten Abschluss der Entwicklung noch Aufwand für die Integration notwendig sein kann. Für agile Projekte ist die Verwendung eines CI-Systems unabdingbar [BW08], aber auch allgemein ist die Verwendung nützlich [Mil08] und vermeidet die „Integration Hell“ [McB00]. Kurze Laufzeiten des CI-Systems motivieren die Entwickler zu häufigerem Übertragen kleiner Änderungen und erhöhen die Bereitschaft zu Code-Refactoring [Bro08]. Beliebte CI-Systeme sind bspw. *Hudson*[6] und das daraus hervorgegangene Programm

[4] https://subversion.apache.org/
[5] http://git-scm.com/
[6] http://hudson-ci.org/

Jenkins[7].

Zur Kommunikation der Softwareentwickler untereinander und ggf. mit den Anwendern und zur Verwaltung anstehender Aufgaben werden insbesondere in umfangreicheren Entwicklungsprojekten *„issue tracker"* [JD03], im Folgenden als Ticketsystem bezeichnet, verwendet. Diese dienen der strukturierten Beschreibung und Bearbeitung von Änderungsanforderungen. So können Fehler und die Begleitumstände erfasst werden, die Relevanz des Fehlers kann beurteilt werden, es sind Verweise zur eingesetzten Programmversion erstellbar und die Bearbeitung durch einzelne Entwickler kann festgehalten werden. Schließlich dient ein Ticketsystem zur Archivierung abgeschlossener Vorgänge. Ticketsysteme können den Umgang mit Fehlern verbessern, allerdings bedeutet die Verwendung eines Ticketsystems selbst einen nicht unerheblichen Aufwand [AHM06]. Insofern ist die Sinnhaftigkeit der Verwendung eines Ticketsystems von Eigenschaften des Entwicklungsprojekts abhängig. Große ggf. verteilte Entwicklerteams und viele Anwender sprechen dafür, kleine Teams und wenige Anwender eher dagegen. Für sie bietet sich die Verwendung von weniger stark strukturierten Systemen, bspw. von Mailinglisten oder kollaborativen Dokumenteneditoren wie *EtherPad*[8], an. Beliebte Ticketsysteme sind bspw. *Bugzilla*[9] und *Redmine*[10].

Wikis dienen der kollaborativen Erstellung und Bearbeitung von Dokumenten. Der bekannteste Einsatz dieser Toolgattung ist die Online-Enzyklopädie *Wikipedia*[11] [EG05]. Ein *Wiki* ist kein spezifisches Tool zur Softwareentwicklung, sondern ein domänenunabhängiges Werkzeug zum Wissensmanagement, das neben dem reinen Vorhalten von Dokumenten auch dazugehörige Funktionen wie Anlegen von Querverweisen, Kategorisieren, Suchen und Protokollieren von Änderungen unterstützt. In der Softwareentwicklung eignen sich Wikis zu Dokumentationszwecken, zur Diskussion und zum Projektmanagement [Lou06]. Ob der Einsatz eines Wikis im Entwicklungsprojekt nützlich ist, lässt sich nicht allgemein konstatieren. Ähnlich wie bei Ticketsystemen bringt der Einsatz eines Wikis selbst Aufwand mit sich, der in kleinen Projekten ggf. den Nutzen übersteigt. In solchen Fällen ist die Verwendung leichtgewichtigerer Methoden zu erwägen. Insbesondere aber, wenn sie als Wissensspeicher über mehrere ähnliche Projekte hinweg verwendet werden, ermöglichen Wikis eine hohe Wiederverwendung von bewährten Artefakten [RBH07]. Bekannte Wiki-

[7] http://jenkins-ci.org/
[8] http://etherpad.org/
[9] http://www.bugzilla.org/
[10] http://www.redmine.org/
[11] https://www.wikipedia.org/

Programme sind TWiki[12] und MediaWiki[13].

Neben diesen alle beteiligten Entwickler betreffenden Komponenten der Infrastruktur existieren noch die individuell von Entwicklern verwendeten Softwareentwicklungswerkzeuge, von denen der Quelltexteditor das zentrale Werkzeug darstellt. Je nach Grad der Integration werden diese Werkzeuge unabhängig voneinander verwendet oder sind als Entwicklungsumgebung (IDE) unter einer gemeinsamen Benutzerschnittstelle verfügbar. Traditionell handelt es sich dabei um lokal installierte Desktop-Software, wobei auch Web-basierte Tools möglich, aber noch nicht weit verbreitet sind [KVKV12]. Während es für weit verbreitete Sprachen wie Java eine große Anzahl von Entwicklungswerkzeugen gibt, ist die Unterstützung von XQuery nicht so weit verbreitet und der Grad der Unterstützung variiert. Editoren, die XQuery umfangreich mit Syntax-Hervorhebung, Code-Vervollständigung, integrierter Dokumentation und Refactoring unterstützen, sind bspw. die proprietär lizenzierten Programme *Altova XMLSpy*[14], *<oXygen/> XML Editor*[15] und das frei lizenzierte, Web-basierte Programm *eXide*[16]. Für andere, weniger XML-spezifische Editoren sind möglicherweise Plug-ins zur die Syntax-Hervorhebung verfügbar, so für *Vim*[17] und NotePad++[18] Die für die beliebten IDEs *Eclipse*[19] und *NetBeans*[20] verfügbaren XQuery-Plug-ins funktionieren derzeit nicht zufriedenstellend. Das *XQDT*[21] genannte Plug-in für Eclipse kann nicht in der aktuellen Eclipse-Version 4.3 installiert werden und die letzten Aktualisierungen stammen aus dem Jahr 2011. Das als *Query XML*[22] bezeichnete Plug-in für NetBeans bietet die Möglichkeit, XQuery innerhalb der IDE auszuführen, verzichtet aber auf wichtige Funktionen wie Syntax-Hervorhebung. Dieses Plug-in funktioniert zwar mit der aktuellen NetBeans-Version 7.4, hat aber seit dem Jahr 2009 keine Pflege mehr erfahren. NetBeans und Eclipse stellen daher bis zur Verfügbarkeit besserer XQuery-Unterstützung keine sinnvollen Werkzeuge für die Entwicklung XML-basierter

[12] http://www.twiki.org/

[13] https://www.mediawiki.org/

[14] http://www.altova.com/xmlspy/xquery-editor.html

[15] http://www.oxygenxml.com/

[16] https://github.com/wolfgangmm/eXide

[17] http://www.vim.org/, Plug-in für XQuery-Syntax unter https://github.com/jeroenp/vim-xquery-syntax.

[18] http://notepad-plus-plus.org/, Plug-in für XQuery-Syntax unter https://github.com/robwhitby/NotepadPlusPlus-XQuery.

[19] http://eclipse.org/

[20] https://netbeans.org/

[21] http://xqdt.org/

[22] http://plugins.netbeans.org/plugin/15704/query-xml

Software dar. Für die IDE *IntelliJ IDEA*[23] steht mit dem Plug-in *IntelliJ XQuery Support*[24] seit Kurzem ein XQuery-Plug-in zur Verfügung, das umfangreiche Unterstützung der Sprache bietet. Bei Entscheidung, welche Softwareentwicklungstools für die Implementierung XML-basierter Software verwendet werden sollen, ist aus technischer Sicht neben der Unterstützung von XQuery und verwandten Sprachen auch die Integration in die übrige Entwicklungsinfrastruktur zu berücksichtigen.

4.4.3 Testfälle

Um die entwickelte Software zu testen, werden auf unterschiedlichen Integrationsstufen Tests benötigt. Damit wird das System hinsichtlich der im Entwurf festgelegten Anforderungen getestet. Funktionale Anforderungen lassen sich relativ unabhängig von der späteren Zielumgebung testen, die Erfüllung nichtfunktionaler Anforderungen, insbesondere an Performanz, hängt jedoch stark von dieser ab. Dass die Zielumgebung im Normalfall nicht vollständig als Entwicklungsumgebung nachgestellt werden kann, muss bei der Interpretation von Performanztests berücksichtigt werden [Mol09]. Statt die absoluten Anforderungen zu messen, kann es in der Entwicklungsumgebung also nötig sein, auf Indikatoren zurückzugreifen. Somit lassen sich auch Performanztests auf den verschiedenen Integrationsstufen durchführen.

Im Folgenden werden Möglichkeiten für Modul-, Integrations- und Systemtests für die hier behandelte XML-basierte Architektur beschrieben.

Modultests Zunächst sind Modultests für einzeln realisierte und testbare Funktionen zu entwickeln. Ein Anhaltspunkt dafür sind die in den Filterschemata des Entwurfs konzipierten Funktionen.

In Lehrbüchern zu XQuery [SWW11b; MB06; LS04] wird das Thema *Software testen* nicht behandelt. Für viele andere Programmiersprachen steht unter dem Begriff *xUnit* [Mes07] eine Klasse von Frameworks für Modultests zur Verfügung, für XQuery besteht noch kein solches plattformübergreifend anerkanntes Framework. XQuery-Interpreter verfügen über eigenständige Lösungen für Modultests, die von der in der XQuery 3.0-Spezifikation vorgesehenen Syntax für *Assertions* Gebrauch machen. Assertions sind im Quelltext explizit ausgedrückte Annahmen über Programmzustände, die zur Laufzeit des Programms ausgewertet werden. Wenn die Assertion verletzt ist, scheitert der Test. Die XQue-

[23] http://www.jetbrains.com/idea/

[24] http://ligasgr.github.io/intellij-xquery/

ry 3.0-Spezifikation beschreibt, wie Assertions aufgebaut sein müssen, legt aber keine Semantik für konkrete Assertions fest. Frameworks für Modultests, die auf Assertions basieren, sind daher implementierungsabhängig, wie *XQSuite*[25] für eXist-db, *xray*[26] für MarkLogic und *Unit Module*[27] für BaseX.

Von der XQuery-Implementierung unabhängig sind die eher leichtgewichtigen Frameworks *xquery-unit*[28], *XQUT*[29] oder selbst implementierte Funktionen für Modultests [Ful08].

Die auf Assertions basierenden Frameworks bieten eine kompaktere Darstellung, während die mit standardisierten XQuery Sprachmitteln implementierten Tests unabhängig von einem konkreten Interpreter funktionieren.

Für XForms ist ein Testframework derzeit noch in Entwicklung [34]. Ob Modultests für XForms angemessen sind, hängt von der Komplexität der Programmlogik in den Formularen ab. Anderenfalls reichen Tests auf der Benutzeroberfläche aus. Ggf. ist für Modultests von XForms derzeit auf selbst entwickelte Tests zurückzugreifen.

Wichtiger als die konkrete Methode, mit der Modultests implementiert werden, ist eine umfangreiche Testabdeckung. Allen Tests ist gemeinsam, dass in jedem Fall eine *Testmethode* und das erwartete *Resultat* festgelegt werden müssen. Sodann müssen die Testfälle zu einer *Suite* zusammengefasst werden, um sie als einzelne Operation aufrufen zu können. Schließlich muss ein Aufruf der Suite einen *Report* produzieren, der die Resultate des Testlaufs enthält [Mes07].

Bei Performanztests auf Ebene der einzelnen Module steht das Messen des Laufzeitverhaltens abhängig von den zu verarbeitenden Daten im Vordergrund, um zu prüfen, ob einzelne Funktionen effizient implementiert sind. Messungen absoluter Zeiten sind auf dieser Ebene mit Ungenauigkeiten behaftet. Daher kann es nötig sein, anstelle von diesen den Code zu modifizieren und bestimmte Operationen zu zählen, um dann das Laufzeitverhalten abhängig von verschiedenen Eingabewerten zu prüfen [Sav07]. Da das Modifizieren von Programmcode zum Zweck des Testens selbst wieder Auswirkungen auf die Qualität haben kann, scheint es geboten, dies nur bei algorithmisch kritischen Funktionen vorzunehmen und ansonsten die Ungenauigkeiten der absoluten Zeitmessung, bedingt durch die Auflösung der Uhr und sonstige Einflüsse auf das Testsystem, durch das ausreichend häufige Wiederholen der Testfälle möglichst zu minimieren.

[25] http://exist-db.org/exist/apps/doc/xqsuite.xml
[26] http://www.xqueryhacker.com/xray/
[27] http://docs.basex.org/wiki/Unit_Module
[28] http://code.google.com/p/xquery-unit/
[29] https://github.com/mblakele/xqut

Integrationstests Um die Kompatibilität der als einzelne Module getesteten Komponenten untereinander sicherzustellen, werden Integrationstests durchgeführt. Sie testen die korrekte Arbeitsweise einzelner Systemfunktionen unter Einbeziehung voneinander abhängiger Komponenten. Diese bilden die als (Sub-)Prozesse entworfenen Systemfunktionen ab.

Technisch lassen sich Integrationstests mit den von Modultests bekannten Werkzeugen umsetzen. Im Unterschied zum Test einzelner Funktionen werden dann voneinander abhängige Funktionen aufgerufen. Wenn bspw. ein Modultest die Funktionen `LadeFormularMetaDaten`, `LadeDaten` und `ErzeugeFormular` aus Programm 4.5 isoliert voneinander und mit fixen Argumenten getestet hat, dann wird eine Testmethode des Integrationstests der Funktion `ErzeugeFormular` die von den vorher ausgeführten Funktionen `LadeFormularMetadaten` und `LadeDaten` zurückgegebenen Daten als Argumente übergeben. Das Resultat bezieht sich dann auf die gesamte, integrierte Funktionalität. Wenn die Tests der funktionalen Anforderungen keine Fehler aufweisen, kann mittels Performanztests die Leistungsfähigkeit der integrierten Komponenten geprüft werden. Dabei wird der Betrieb mit der maximal zu erwartenden Last simuliert und Kennzahlen wie Systemantwortzeiten werden gemessen. Bei der Definition der Last und der Interpretation der Ergebnisse ist zu berücksichtigen, dass sich die Leistungsfähigkeit der Hardware in der Testumgebung ggf. von der des Zielsystems unterscheidet.

Eine umfassende Beschreibung von Integrationstest liefert [WES+13].

Systemtests Systemtests testen das integrierte System aus Sicht der geplanten Verwendung. Es wird die vollständige Funktionalität in einer Umgebung, die der späteren Produktionsumgebung möglichst ähnlich ist, getestet. Es wird getestet, ob das System die in der Analysephase definierten Prozesse korrekt ausführt. Dabei werden die vorgesehenen Schnittstellen benutzt. Da hierbei keine Implementierungsdetails relevant sind, werden keine XQuery- oder XForms-spezifischen Tools benötigt.

Für die Tests der Benutzeroberflächen stehen Tools bereit, die die Eingaben eines menschlichen Benutzers ausführen. *Selenium*[30] ist ein Framework, um Browseraktivität zu automatisieren. Für Systemtests werden damit Testfälle, die Benutzerinteraktionen auf der Weboberfläche durchführen, ausgeführt. Das Resultat der Interaktionen wird über die Weboberfläche abgefragt und mit dem erwarteten Resultat verglichen. Ausgehend von der Implementierung in Selenium ist derzeit mit *WebDriver* [SB13] ein W3C-Standard zur Browsersteuerung

[30] http://docs.seleniumhq.org/

in Entwicklung, so dass das Testen mit unterschiedlichen Browsern vereinfacht wird.

Neben den über die Benutzeroberfläche ausgelösten Prozessen sind die per Webservice aufrufbaren Interaktionen mit Maschinen ebenfalls zu testen. Dafür werden die Anfragen an die implementierten Webservices durch Testfälle ersetzt. Ein Tool, das dies sowohl für SOAP- als auch REST-basierte Webservices leistet, ist bspw. *SoapUI*[31].

Wenn die funktionalen Systemtests fehlerfrei sind, können Performanztests in der Testumgebung durchgeführt werden und so eine Einschätzung der Leistungsfähigkeit des Gesamtsystems unter simulierten Benutzereingaben und Interaktionen mit angeschlossenen Systemen ermöglichen. Auch hierbei sind die Auswirkungen unterschiedlich leistungsfähiger Hardware in Testumgebung und Zielsystem zu beachten.

Eine umfangreiche Darstellung der Aspekte von Systemtests ist in [SBS12] zu finden.

Abnahmetest Der am Ende der Testaktivitäten stehende Abnahmetest findet nicht durch die Entwickler sondern durch die Verwender der Software statt und bildet den Übergang zur Einführung. Hierbei prüft der Abnehmer, ob die Software die Anforderungen erfüllt. Da ein vollständiger Test nicht möglich ist, ist umso wichtiger, dass der Abnehmer sich davon überzeugt, dass der Softwareentwicklungsprozess korrekt durchgeführt wurde [Bal11]. Die Reports aus der Durchführung der Tests dokumentieren einen wichtigen Bestandteil der korrekten Entwicklung.

4.4.4 Programmierung

Durch die Programmierung erfolgt die Implementierung des Programmcodes. Dazu werden im Folgenden zunächst allgemeine Prinzipien der Programmierung betrachtet. Sodann werden XML Schema-spezifische Überlegungen für ein gutes Design der Datenhaltungsschicht angestellt. Für eine robuste Implementierung von XQuery-Modulen werden Entwurfsmuster betrachtet und schließlich folgen noch Hinweise zur Erstellung der Benutzeroberfläche mit XForms.

Prinzipien der Implementierung Unabhängig von der verwendeten Technik werden von Balzert für die Implementierung von Software die folgende Prinzipien empfohlen:

[31] http://www.soapui.org/

- „Prinzip der Verbalisierung
- Prinzip der problemadäquaten Datentypen
- Prinzip der Verfeinerung
- Prinzip der integrierten Dokumentation“ [Bal01]

Unter *Verbalisierung* wird verstanden, den Programmcode so zu schreiben, dass die ihm zugrundeliegenden Ideen und Konzepte in Worten ausgedrückt im Programm sichtbar sind. Dazu gehören die Benennung von Programmkonstrukten wie Variablen und Operationen mit aussagekräftigen, problembezogenen Bezeichnern und das Explizieren von Konstanten. Ebenso gehören geeignete Kommentare zur Verbalisierung des Programmcodes. Gute Kommentare stellen die Intention des Codes dar, ohne die durch den Programmcode selbst bereits ausgedrückte Funktionsweise zu wiederholen [McC93]. Die Formatierung des Programmcodes orientiert sich an der Schachtelung von FLWOR-Ausdrücken und geschieht durch Einrückungen des Quelltextes. Weitere Hinweise finden sich z. B. in *XQuery Style Conventions* [McB06].

Die Forderung nach *problemadäquaten Datentypen* bedeutet, dass die seitens fachlicher Anforderungen gegebene Domäne der Daten möglichst präzise durch die Verwendung von Datentypen abgebildet wird. Hierdurch werden die Verständlichkeit des Programmcodes erhöht und Typprüfungen ermöglicht. Adäquate Datentypen für XML-Dokumente werden durch ein geeignetes XML Schema festgelegt und können in XQuery durch Import des Schemas verwendet werden. Falls kein Schema importiert werden kann, etwa wenn Funktionen für wiederverwendbare Bibliotheken realisiert werden, kann auf die mitgelieferten Datentypen des *XQuery and XPath Data Model* zurückgegriffen werden.

Das Prinzip der *Verfeinerung* fordert, Programmcode durch unterschiedliche Abstraktionsstufen zu strukturieren und dies durch Kommentierung deutlich zu machen. So werden Kommentare auf oberster Ebene, die eine kompakte Darstellung des Programmcodes enthalten, weiter bis auf Detailebene verfeinert. Dadurch wird der Entwicklungsprozess des Programmcodes dokumentiert und die Entwicklung kann von Dritten nachvollzogen werden.

Das Prinzip der *integrierten Dokumentation* umfasst neben der Kommentierung des Programmcodes auch die Angabe weiterer Informationen. So bieten Dokumentationskommentare, in XQuery mittels xqDoc [26], die Möglichkeit, eine Programmdokumentation aus Quelltextkommentaren zu erstellen, indem der Quelltext mit Annotationen versehen wird, aus denen ein Compiler automatisiert die Dokumentation erzeugt.

Programm 4.4 zeigt ein Beispiel für die Anwendung der genannten Prinzipien. In die Zeichen (: und :) eingeschlossene Kommentare enthalten gemäß dem Prinzip der Verfeinerung Informationen zur internen Funktionsweise des Programmcodes, wobei sie mit steigender Detaillierung des Programms, erkennbar durch die Einrückung, ebenfalls Bezug auf die Implementierungsdetails nehmen. Die integrierte Dokumentation verwendet die Dokumentationskommentare von xqDoc. Diese werden durch die Zeichen (:~ eingeleitet und enthalten die nach außen relevante Beschreibung des Moduls und der Funktion. Die mit dem Zeichen @ beginnenden Annotationen zeichnen Inhalte semantisch aus und werden durch den Dokumentationsgenerator ausgewertet. Bei der Modulbeschreibung sind dies die Informationen zu Version und Autor, in der Dokumentation der Funktion `lib:generate-role-nodes` werden die Argumente und die Rückgabe besonders hervorhoben. Eine aus diesem Codebeispiel mittels xqDoc generierte Dokumentation ist in Abbildung 4.11 gezeigt. Die Bezeichnung der Variablen und der Funktion im Codebeispiel folgt dem Prinzip der Verbalisierung, indem sprechende Bezeichner verwendet werden. Die Funktionssignatur folgt dem Prinzip der problemadäquaten Datentypen, indem die Datentypen explizit deklariert werden, sodass Fehler in Bezug auf Eingabe- oder Ausgabetypen erkannt werden. So muss der Funktion ein Parameter vom Typ Element mit dem Namen `odm:AdminData` übergeben werden, und der Rückgabewert ist ein Element namens `odm:RoleData`.

Versionierung wird von Balzert [Bal01] ebenfalls als Teil der integrierten Dokumentation gesehen. Bei Verwendung von Codeverwaltungssystemen erscheint es allerdings ratsam, zwei Arten von Versionen zu unterscheiden: Die interne Codeversionierung, die bei Verwendung eines Code-Repositorys automatisch jeder einzelnen Übertragung in das Repository zugewiesen wird, und die externe Versionierung, die sich auf veröffentlichte Software bezieht.

Die interne Versionierung wird durch die Technik des Repositorys bestimmt und besteht bspw. aus einer fortlaufenden Nummer oder Hashwerten. Sie dient den Entwicklern der Software zum Identifizieren unterschiedlicher Codeversionen. Diese interne Versionsnummer findet sich meist nicht im Programmcode, da sie durch das Repository generiert und von diesem verwaltet wird.

Die externe Versionierung wird explizit vorgenommen und richtet sich an die Nutzer der Software. Der Versionsstand von veröffentlichter Software wird gemäß einem Schema gebildet, bspw. bei Verwendung des *Semantic Versioning* [29] in der Form X.Y.Z, wobei die Buchstaben ganzzahlige, nicht-negative Werte annehmen. X bezeichnet dabei die Hauptversion, Y die Unterversion und X das Patchlevel. Die Änderung dieser Werte hat eine festgelegte Semantik: Die

```
1  xquery version ”3.0”;
2  (:~
3   : This module contains transformation functions for ODM documents
4   : @see http://www.cdisc.org/odm
5   : @author Christian Forster
6   : @version 0.1.0
7   :)
8  module namespace lib=”http://example.org/lib”;
9
10 import schema namespace odm=”http://www.cdisc.org/ns/odm/v1.3” at ”x4T-
       ODM1-3-1.xsd”;
11
12 (:~
13  : Sorts ODM user elements according to their role and creates a structure
        like <RoleData><Role Type=”FirstRole”><User OID=”1”/><User OID=”2”/
       ></Role><Role Type=”SecondRole”><User OID=”3”/><User OID=”4”/></Role
       ></RoleData>
14  :
15  : @param $doc the document that contains the user elements
16  : @return user elements grouped by role
17  :)
18 (:
19  : Creates an element RoleType for every UserType that is present in the
       document. Users are inserted into their respective RoleType element.
20  :)
21 declare function lib:generate-role-nodes($doc as element(odm:AdminData))
       as element(odm:RoleData)
22 {
23   (: Create root element :)
24   <RoleData xmlns=”http://www.cdisc.org/ns/odm/v1.3”>{
25     let $originalUsers as element()* := $doc/odm:User
26     (: Identify Roles and iterate :)
27     for $role in distinct-values($originalUsers/@UserType/string())
28     order by $role
29     return
30       <Role Type=”{$role}”>{
31       (: Identify User with current role :)
32        for $processedUser in $originalUsers
33         where $processedUser/@UserType/string() = $role
34         return
35           (: Create an element User with all attributes but UserType and
                 all child elements from the originating User element :)
36           <User>{ ($processedUser/@*[name() != ”UserType”], $processedUser
                 /*) }</User>
37       }</Role>
38   }</RoleData>
39 };
```

Programm 4.4: Ein Softwarebibliotheksmodul, das entsprechend den Implementierungsprinzipien programmiert ist.

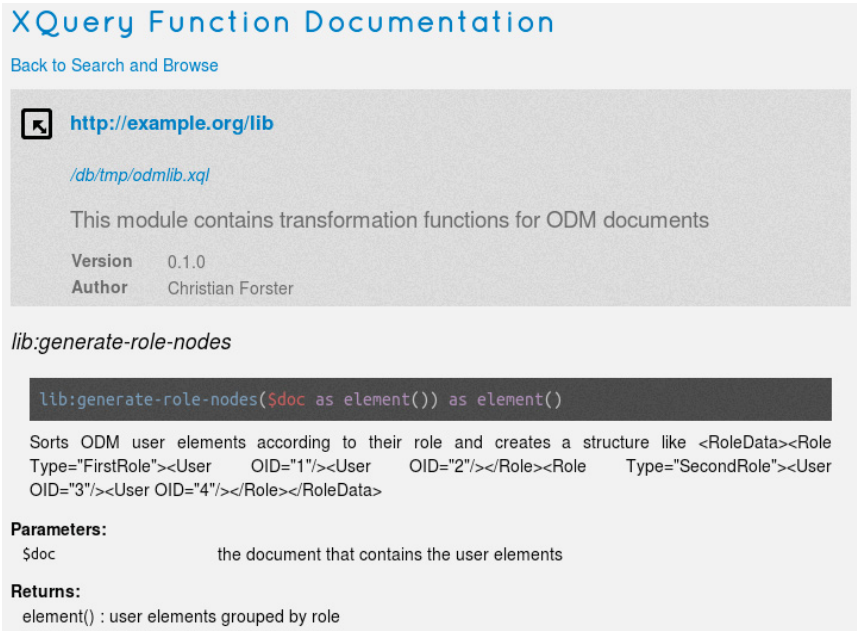


Abbildung 4.11: Mittels xqDoc erzeugte Dokumentation des Moduls aus Programm 4.4.

Entwicklung startet mit Version 0.1.0, die auch das Programm 4.4 aufweist. Der erste als stabil veröffentlichte Softwarestand erhält die Version 1.0.0. Finden Änderungen statt, die nur der Behebung von internen Softwarefehlern dienen, so wird das Patchlevel inkrementiert. Änderungen, die neue Funktionen hinzufügen, aber die Kompatibilität der Software nicht beeinträchtigen, führen zu einer Erhöhung der Unterversion. Bei Änderungen, die nicht zu vorherigen Versionen kompatibel sind, wird die Hauptversion erhöht. Bei jeder Änderung der Unterversion wird das Patchlevel auf den Wert 0 zurückgesetzt, bei der Änderung der Hauptversion geschieht dies für Unterversion und Patchlevel. So kann der Anwender anhand der Versionsnummer einordnen, welche Art von Änderungen bei einem Update erfolgt ist. Bei APIs und Bibliotheken kann auf Grundlage des Semantic Versioning eine automatisierte Abhängigkeitsprüfung erfolgen.

Indizierung der XML Schema Indizes sind eine Methode, um Datenzugriffe zu beschleunigen, indem der Weg zum Speicherort der Daten durch Berechnungen oder Abkürzungspointer vereinfacht wird [Vos08]. Bei XML Indizes sind zwei Arten zu unterscheiden: Indizes für *strukturelle* und *wertbasierte* Bestandteile der Anfragen [CVZ+02]. Strukturelle Vergleiche suchen anhand der XML-Struktur nach dem Ziel der Anfrage, während wertbasierte nach dem Inhalt des Zielelementes suchen. Oft kommen beide Arten in einer Anfrage vor, wie in der folgenden:

```
doc("data.xml")/ODM//User[FirstName = "Petra"]
```

Die in eckigen Klammern stehende Bedingung gibt an, dass ein Vergleich mit dem Elementinhalt auf den Wert `Petra` vorgenommen werden muss. Wenn ein wertbasierter Index mit den Werten der Elemente `FirstName` zur Verfügung steht, so müssen nicht sämtliche dieser Elemente gesucht und verglichen werden, sondern das oder die passenden Elemente können direkt aufgefunden werden. Je nach XML-DBMS können weitere Indizes konfiguriert werden. Es können etwa die Werte als Volltexte indiziert sein, um leichter nach einzelnen Worten suchen zu können oder als Bereichsindizes, die bei sortierbaren Datentypen nach Werten aus bestimmen Intervallen suchen können. Der restliche Bestandteil der Anfrage gibt an, an welcher Stelle der XML-Struktur das zu suchende Element `User` zu finden ist, nämlich unterhalb des Elements `ODM`. Ohne strukturellen Index müssten alle Elemente, die sich hierarchisch unterhalb des Elementes `ODM` befinden untersucht werden. Durch einen strukturellen Index kann diese Suche abgekürzt werden, da bekannt ist, wo sich die Elemente `User` befinden.

Die Verfügbarkeit und Definition der Indizes ist von der Verwendung des jeweiligen XML-DBMS abhängig. Ein Vorschlag für einen Algorithmus zur automatisierten Indexerzeugung basierend auf der Analyse des Workloads, des Schemas und statistischer Eigenschaften der Daten findet sich in [RPBP04]. In praxi erzeugen XML-DBMS allerdings nur bestimmte, meist strukturelle Indizes automatisiert für alle XML-Dokumente, während wertbasierte Indizes manuell konfiguriert werden müssen. Dabei richtet sich die Frage, welche Indizes angelegt werden sollen, nach dem Nutzungsprofil der Anwendung. Formal handelt es sich bei der Auswahl der optimalen Indizes bei relationalen Daten um ein NP-vollständiges Problem [Pia83], für XML-Dokumente gilt dies ebenfalls [Ham04]. Eine Lösung bedarf daher einer Heuristik, die aus der geplanten oder beobachteten Nutzung der Datenbank sinnvolle Kandidaten für Indizes identifiziert. Es lässt sich festhalten, dass Indizes Leseoperationen beschleunigen, da die Daten schneller gefunden werden, aber bei schreibenden Operationen eine höhere

Last verursachen, da die Indexstruktur beim Schreibzugriff angepasst werden muss [LSSS07]. Für die Implementierung lässt sich daraus ableiten, dass für Elemente, die häufig gesucht werden, ein Index für die entsprechende Zugriffsart angelegt werden sollte. Durch Tests und spätere Beobachtung des Echtbetriebs kann erkannt werden, welche Anfragen ggf. durch Modifikation der Indizes optimierbar sind. Die derzeit bekannten und produktiv einsetzbaren XML-DBMS bieten noch keine selbstständige Indexoptimierung.

XQuery Module und Entwurfsmuster Zentral für diese Phase ist die Implementierung der entworfenen Funktionen als XQuery Module, die als Geschäftslogik von Benutzeroberfläche und Webservice-Schnittstellen verwendet werden.

Für objektorientierte Programmiersprachen existiert eine Reihe von mittlerweilen in der Praxis etablierten Entwurfsmustern, die für wiederkehrende Anforderungen in der Softwareentwicklung einen standardisierten Lösungsweg aufzeigen [GHJV94]. Sie dienen dazu, eine naive Implementierung durch ein erprobtes Muster zu ersetzen und so die Qualität des Programmcodes zu erhöhen. Diese objektorientierten Entwurfsmuster sind nicht bedingungslos in funktionalen Sprachen anwendbar. Einige Muster benötigen zwingend objektorientierte Eigenschaften, wie das Muster *Kompositum*, das ohne Vererbung nicht funktioniert, und das Muster *Besucher*, das auf das Überladen von Methoden aufbaut [AH02]. Das Muster *Fassade* hingegen fasst mehrere Schnittstellen zu einer einzigen zusammen und ist nicht auf objektorientierte Spracheigenschaften angewiesen [AH02]. Zum Teil bestehen auch die Probleme, die durch ein objektorientiertes Entwurfsmuster gelöst werden, in funktionalen Sprachen nicht: Da bspw. Funktionen höherer Ordnung verwendet werden können, entfällt die Notwendigkeit das Entwurfsmuster *Befehl* in funktionalen Sprachen zu implementieren [18].

Eine Kategorisierung von objektorientierten Entwurfsmustern hinsichtlich ihrer Eignung für generische modulare Programmierung, funktionale Programmierung und rein objektorientierte Programmierung findet sich in [Nar08]. Da XQuery funktionale und modulare Eigenschaften hat, erscheinen Implementierungen von Entwurfsmustern auf dieser Grundlage möglich.

Die Umsetzung der Entwurfsmuster *Strategie*, *Beobachter* und *Zuständigkeitskette* [GHJV94] ist in XQuery möglich [CBK10]. Dabei werden die Intention es jeweiligen objektorientierten Entwurfsmusters genannt, eine Implementierung in XQuery angedeutet und der Ansatz diskutiert. Das Ziel dieser Entwurfsmuster ist, den Programmcode hinsichtlich Wiederverwendbarkeit, loser Koppelung und bzw. oder Flexibilität gegenüber Änderungen zu verbessern.

Außerdem beschreibt [CBK10] ein neues, „Translator“ genanntes Muster für XQuery. Es dient der einfacheren Bearbeitung heterogen strukturierter XML Dokumente, indem der von XSLT bekannte Transformationsstil mit XQuery nachgebildet wird, und wird im Folgenden aufgrund seiner Neuartigkeit erläutert. Eine naive Implementierung zur Transformation, wie Programm A.3 in Abschnitt A.2 im Anhang zeigt, iteriert mittels eines FLWOR-Konstrukts über die Elemente des Quelldokumentes und wendet für jedes Element einen spezifischen Transformationsausdruck an. Somit weisen die Ablaufsteuerung und alle Templates eine hohe Abhängigkeit untereinander auf.

Durch Anwendung des Entwurfsmusters Translator werden die unterschiedlichen Transformationsfunktionen für die heterogenen Bestandteile des Quelldokuments in voneinander unabhängige Templates ausgelagert, wodurch die Koppelung der Module sinkt. Templates stellen alle Funktionen bereit, um Elemente des Quelldokuments in Elemente des Zieldokuments umzuwandeln. Das Muster nutzt Funktionen höherer Ordnung, um diese Transformationsfunktionen als Argumente an die Ablaufsteuerung zu übergeben.

Das Muster ist bspw. geeignet, um Personendaten, die in unterschiedlichen Formatierungen empfangen werden, in ein einheitliches Zielformat zu transformieren. Das Sequenzdiagramm in Abbildung 4.12 zeigt den Ablauf. Zentrale Rolle spielt die Funktion `transform` zur Ablaufsteuerung. Sie wird vom Client aufgerufen und wendet die Templates auf das Quelldokument an. Als Argumente erhält sie das Quelldokument und eine Sequenz von Funktionen, die die anzuwendenden Templates zurückgeben. Die Funktion `transform` iteriert über die Elemente des Quelldokuments und wendet die Funktionen der Templates an: Mit einer Funktion `match-`*`Elementname`* wird geprüft, welche Transformationsfunktion anzuwenden ist. Zu jeder `match`-Funktion enthält das Template eine Funktion mit der Bezeichnung `apply-`*`Elementname`*, die die Transformation für ein einzelnes Element ausführt. Sollen die Kindelemente ebenfalls transformiert werden, so erfolgt ein weiterer Aufruf der `transform`-Funktion innerhalb der `apply`-Funktion. Eine vollständige Implementierung inklusive eines Quelldokuments und dem dazugehörigen Zieldokument befindet sich in Abschnitt A.2. Das dort gezeigte Beispiel verwendet Daten in den Formaten ODM und xCard [Per11b], einer XML-basierten Version des zum Adressaustausch verbreiteten Formats vCard [Per11a].

Das Hinzufügen neuer Templates oder Ändern bestehender wird bei Verwendung des Musters erheblich vereinfacht, da Modifikationen nur das jeweilige Modul betreffen. Ebenso ist die Wiederverwendung einzelner Templates unabhängig von anderen möglich.

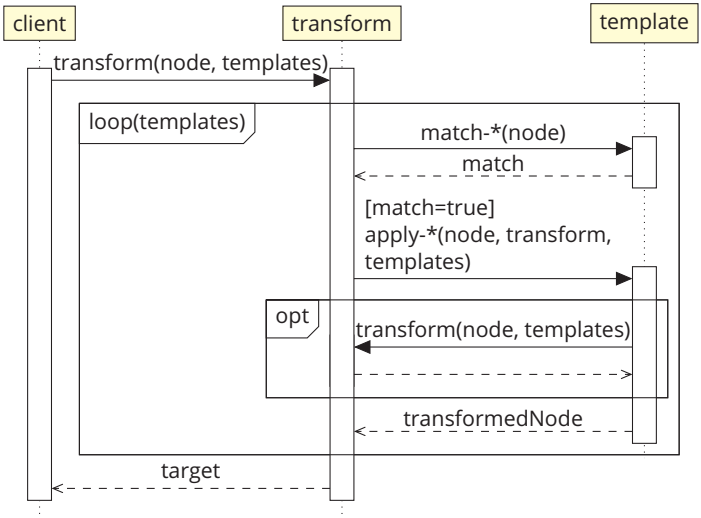


Abbildung 4.12: Funktionsaufrufe im Entwurfsmuster Translator, vgl. [CBK10].

In [CBK10] wird davon ausgegangen, dass neben den vorgestellten XQuery Entwurfsmustern noch weitere existieren und eine systematische Zusammenstellung angekündigt[32]. Durch die Erprobung und Bewährung in der Praxis lassen sich aus Implementierungen Entwurfsmuster extrahieren, sie stellen also eine ex post Dokumentation dar [GHJV94]. Aus diesem Grund lassen sie sich nicht rein theoretisch herleiten. Es ist aber plausibel, die Ziele von Entwurfsmustern, nämlich gut strukturierten und somit wartbaren und wiederverwendbaren Code, beim Implementieren von Software zu beachten.

Implementierung der Oberfläche Für die Implementierung der datenbasierten Bestandteile der Benutzeroberfläche spielt XForms die zentrale Rolle. Die übrigen Bestandteile der Oberfläche werden mittels herkömmlicher Webtechniken wie HTML, JavaScript und CSS implementiert, wozu umfangreiche und leicht auffindbare Literatur zur Verfügung steht.

Die Implementierung der XForms geschieht gemäß der Aufteilung des MVC-Musters. Das *Model* wird durch die Repräsentation der XML-Datenstruktur gebildet. Die *View* wird durch die Komponenten der Benutzerinteraktion gebildet.

[32]Unter der dafür angegebenen URL http://patterns.28msec.com/ ist im Januar 2014 allerdings nichts hinterlegt.

Der *Controller* bindet die Komponenten aneinander, indem er auf Events reagiert.

Bei der Implementierung eines Formulars stellt sich die Frage, ob es generativ erzeugt oder direkt implementiert werden soll. Für die generative Erzeugung werden Regeln, bspw. in XSLT, festgelegt, die für ein XML-Dokument, das als Model dient, die View- und Controller-Komponenten zur Laufzeit ableiten. Bei der direkten Implementierung werden die Komponenten für eine Dokumentstruktur durch den Programmierer erstellt.

Die direkte Implementierung kann verwendet werden, wenn der Aufbau aller als Model zu bearbeitenden Dokumente ähnlich ist. Dies ist bspw. beim Formular zum Anlegen von Nutzern, wie in Abschnitt 3.3 gezeigt, der Fall. Hierbei ist keine genau gleiche Struktur nötig, Variationen, die auf der Wiederholung von Elementen basieren, können durch das XForms-Element `repeat` abgebildet werden. Falls z. B. mehrere E-Mail-Adressen für einen Nutzer verwendbar sein sollen, so kann dies auch durch direkte Implementierung realisiert werden.

Die generative Erzeugung ist nötig, wenn das Schema bekannt, der tatsächliche Aufbau der XML-Dokumente aber a priori unbekannt ist. Außerdem kann sie helfen, ähnliche Strukturen nicht redundant zu programmieren, sondern per Template zu generieren [Vli13a]. Die generative Erzeugung kommt bspw. für die Studiendokumentationsformulare zur Anwendung. Die Metadaten werden konform zum XML Schema des ODM-Formats importiert, der Aufbau der Studiendokumentation ist aber für jede Studie unterschiedlich. Dieser ergibt sich aus den Elementen unter `odm:MetaDataVersion`, während die tatsächlichen Daten für einen Studienteilnehmer in der Struktur `odm:SubjectData` gespeichert werden. Es wird zuerst ein Stylesheet erstellt, das Templates für die Elemente der Metadaten enthält, um initial ein XML-Dokument zu generieren, das persistiert und als Model verwendet wird. Ein weiteres Stylesheet dient zur Erzeugung des vollständigen XForms-Codes. Es bindet das generierte Model ein und erzeugt View und Controller. Integritätsbedingungen, die aus den Metadaten hervorgehen, wie die Bindung von Datentypen an Elemente des Models, werden ebenfalls damit in XForms umgesetzt. Abbildung 4.13 zeigt den Ablauf.

Die Implementierung der entworfenen Benutzeroberfläche geschieht unter Verwendung der von XForms spezifizierten Elemente. Die Literatur zu XForms beschreibt die Verwendung dieser anhand von Anwendungsbeispielen, von denen einige als Entwurfsmuster gelten können.

In [Dub03] werden Strategien zur Verbindung von Model und View vorgestellt. Bei Verwendung des Musters „Stepwise XPath“ entsteht eine direkte, einfache Referenzierung der Elemente der View auf die des Models. Dabei entsteht

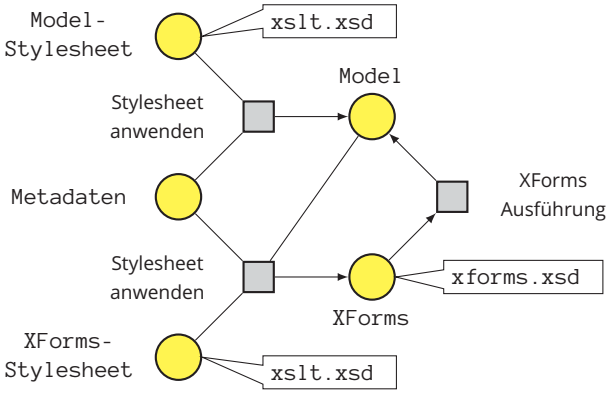


Abbildung 4.13: Generative Erzeugung eines XForms-Formulars zur Laufzeit.

allerdings eine stärkere Bindung als durch das Muster „Design by Buddy“. Bei diesem Muster werden die Elemente des Models an explizite Identifikatoren gebunden. Somit können Model und View getrennt voneinander entwickelt werden, nachdem die Bindungen als Schnittstelle festgelegt worden sind. Das Muster „XML Localisation“ beschreibt, wie Mehrsprachigkeit in XForms erreicht werden kann.

Das Wikibook zu XForms [38] enthält viele Anwendungsbeispiele, von denen einige als „Best Practice“ markiert sind, da sie von mehreren Autoren für einen angemessenen Lösungsweg gehalten werden. Diese reichen von der eher trivialen Eingabe von Passwörtern bis zur Realisierung von Navigationsmenüs für komplexe Formulare.

In weiteren Webquellen und Herstellerdokumentationen sind ebenfalls Anleitungen zur Implementierung von wiederkehrenden Mustern zu finden, bspw. zur verketteten Auslösung von Aktionen als „submission chain“ [33] oder zum dynamischen Laden von Vorschlägen in Suchfeldern [12].

4.5 Einführung und Nutzung

Die in der Implementierungsphase erstellte und getestete Software wird in der Produktionsumgebung eingeführt und anschließend benutzt. Während des Betriebs der Software treten eventuell Fehler auf, externe Anforderungen ändern sich oder neue Anforderungen entstehen. Dies macht Wartungs- und Pflegeakti-

vitäten nötig. Diese Schritte unterscheiden sich bei der Verwendung der hier behandelten XML-basierten Architektur nicht von denen mittels sonstiger Technik realisierten Webanwendungen und werden daher im Folgenden nur kurz wiedergegeben.

4.5.1 Systemeinführung und Betrieb

Nach Durchführung des Abnahmetests kann die Software neu in Betrieb genommen bzw. eine Aktualisierung der bestehenden Version durchgeführt werden. Insbesondere bei Neuentwicklung der Software müssen die Benutzer entsprechend geschult worden sein und ein Umstellungsplan, der angibt, welche Tätigkeiten bei der Einführung zu leisten sind, muss erstellt worden sein [SH05]. Wichtig ist, für eine eventuell nötige Übernahme von Daten aus Vorgängersystemen eine Datenmigration zu planen. Die Einführung selbst kann schlagartig zu einem Stichtag mit Ablösung des oder der Vorgängersysteme oder stufenweise mit einem Teil des Systems, des Benutzerkreises oder der Daten und damit verbundenem Parallelbetrieb mit dem oder den Vorgängersystemen erfolgen. Das letztgenannte Vorgehen hat das Ziel, Umstellungsrisiken zu mindern, erzeugt aber mehr Aufwand für die Pflege der Daten [SH05].

Bei Verwendung eines agilen Entwicklungsmodells wird häufig eine neue Softwareversion mit kleineren Änderungen eingeführt. Damit die Entwickler über von den Anwendern gewünschte Änderungen zeitnah erfahren können, sollten die Benutzer über eine Kommunikationsmöglichkeit mit den Entwicklern verfügen, z. B. per Zugriff auf das Ticketsystem. Neben den Benutzern können auch von Systemadministratoren, die Logfiles des Systems und Betriebseigenschaften wie Speicherbedarf und Prozessorauslastung einsehen können, Hinweise auf Probleme erfolgen.

4.5.2 Wartung und Pflege

Änderungswünsche der Anwender und Fehler des Systems gehen als neue Anforderungen in die Analysephase ein, womit sich der Softwareentwicklungszyklus schließt. Je nach Art der Änderung wird zwischen *Wartung* und *Pflege* unterschieden [Bal11].

Wartungsarbeiten beheben dabei Fehlerursachen in der Software und verfolgen die Ziele *Stabilisierung und Korrektur* sowie *Optimierung*. Stabilisierung und Korrektur zielen darauf ab, Fehler, die während der Testphase nicht entdeckt worden sind und erst im Betrieb auftreten, zu beheben. Optimierung erhöht die

Effizienz der funktional korrekten Software, indem sie Änderungen umfasst, die den Ressourcenbedarf senken oder zu Geschwindigkeitssteigerungen führen.

Pflegetätigkeiten führen Änderungen und Erweiterungen an einem bislang zufriedenstellend arbeitenden Softwaresystem durch. *Anpassungen und Änderungen* sind dabei die Tätigkeiten, die durch geänderte externe Bedingungen nötig werden, bspw. Modifikationen aufgrund von Änderungen der Laufzeitumgebung oder gesetzlicher Vorgaben. *Erweiterungen* fügen der Software neue Funktionen hinzu.

4.6 Zwischenfazit

Die hier gezeigte XML-basierte Architektur kann mittels klassischer und agiler Methoden entwickelt werden. Zur agilen Entwicklung, bei der Teilfunktionen der Software nach Nützlichkeit für den Anwender priorisiert erstellt und ausgeliefert werden, passt der Verzicht auf Transformationskomponenten in der Architektur, die Daten in andere Formate konvertieren, aber keine aus fachlicher Sicht Nutzen stiftende Funktion haben.

Die Werkzeuge für Analyse, Entwurf und Implementierung unterscheiden sich teilweise von denen, die bei der Entwicklung von Software nach der traditionellen Schichtenarchitektur zum Einsatz kommen. XML-Netze spielen in der Modellierung eine wichtige Rolle und weisen in Kombination mit der hier vorgestellten XML-basierten Architektur den Vorteil auf, dass deren Datenmodelle bereits XML-basiert sind und in die Implementierungsphase übernommen werden können. Die besondere Einfachheit der Architektur entsteht durch den transformationsfreien Übergang zwischen den einzelnen Komponenten und die Verwendung eines einheitlichen Datenmodells, das ggf. in Teilen auf domänenspezifisch standardisiertem XML Schema beruht. Für die Implementierung benötigte XML-DBMS, XQuery-Implementierungen und XForms-Interpreter sind im Jahr 2014 weniger weit verbreitet als relationale DBMS, objektorientierte Programmiersprachen und Frameworks sowie dazu passende HTML-basierte Darstellungskomponenten, was sich als Einschränkung bei der Verfügbarkeit von Entwicklern und Dienstleistungen erweisen kann. Wartung und Pflege der XML-basierten Architektur erfordern kein grundlegend neues Vorgehen, die Architektur erspart allerdings die ansonsten nötige Wartung der Transformationskomponenten.

5 Implementierung einer XML-basierten Webanwendung

Das Kapitel zeigt ein Gerüst für eine XML-basierte Webanwendung. Es werden im Rahmen der vorgestellten Architektur wiederverwendbare Komponenten als Grundlage für die fachlichen Funktionen implementiert. Aus Sicht der Prozessorientierung sind dies Subprozesse oder einzelne Aktivitäten, die von Geschäftsprozessen verwendet werden können. Die Anwendung soll Routing nach dem MVC-Muster implementieren, Benutzer und deren Zugriffsrechte verwalten, mit Webservices kommunizieren und Softwaretests unterstützen. Die sodann entwickelte Anwendung x4T baut auf den hier präsentierten Komponenten auf, indem die domänenspezifischen Anforderungen hinzugenommen werden. Die x4T-Software wurde im Rahmen eines Forschungsprojekts zu Nutzung von klinischer Behandlungsdokumentation in der klinischen Forschung entwickelt. Bei der Durchführung klinischer Studien soll die teilweise redundante Erfassung von Daten, die der derzeitigen Praxis entspricht, zugunsten einer einmaligen Erfassung und erleichterten Wiederverwendung vermieden werden. Die Anforderungen an diese Anwendung und ihr Konzept werden zunächst erläutert. Sodann werden die Benutzer und Programm-Schnittstellen vorgestellt, schließlich wird auf die Umsetzung eingegangen.

5.1 Domänenunabhängiges Gerüst

5.1.1 MVC

Controller

Jede HTTP-Anfrage wird zunächst vom Controller entgegengenommen. Diese zentrale Komponente entscheidet, welche Teile der Anwendung für die Bearbeitung zuständig sind.

Analyse Der eingehende Request wird analysiert, es wird geprüft, welche Funktion angefordert wird und ob der anfragende Nutzer über die dafür notwendige

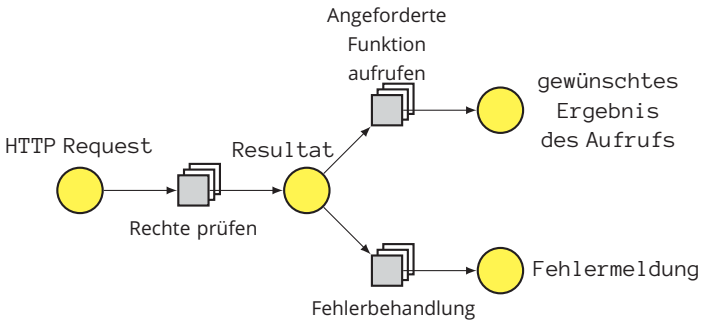


Abbildung 5.1: Analyse der Anfragebearbeitung im Controller.

Berechtigung verfügt. Dazu nutzt der Controller das Rechtemanagement und erhält ein Resultat dieser Überprüfung. Auf Grundlage dessen wird entschieden, wie die Anfrage beantwortet wird. Bei ausreichenden Rechten übergibt der Controller an die für die Bearbeitung zuständige View-Komponente. Bei nicht ausreichenden Rechten wird eine Fehlermeldung zurückgegeben. Diese kann mit der Aufforderung eines Logins als berechtigter Nutzer verknüpft werden. Abbildung 5.1 zeigt den Vorgang.

Entwurf Jeder HTTP-Request durchläuft in der Webanwendung einen zentralen Controller. Dieser entscheidet, durch welche Komponente der View die Anfrage bearbeitet wird. Zunächst wird in diesem Prozess geprüft, ob die Anfrage den Login eines Benutzer erfordert. Falls nicht, übergibt der Controller den Request direkt an die Komponente. Falls ein Benutzer erforderlich ist, wird geprüft, ob eine Session vorhanden ist. Falls nicht, wird der Request auf Anmeldedaten untersucht oder der Benutzer zum Login aufgefordert. Technisch geschieht die Einbettung des Logins durch eine Umleitung auf eine Anmeldeseite und eine anschließende Wiederholung des ursprünglichen Requests. Sodann wird geprüft, ob der in der Session angemeldete Nutzer über die notwendigen Rechte zur Ausführung der gestellten Anfrage verfügt. Hierzu kann das in Abschnitt 4.3.3 besprochene Modell des attributbasierten Rechtemanagements verwendet werden, in einfacheren Fällen auch ein Rollenmodell. Nachdem das Ergebnis der Berechtigungsprüfung bekannt ist, wird das Resultat der Berechtigungsprüfung an den Controller zurückgegeben. Abbildung 5.2 zeigt den Prozess der Berechtigungsprüfung. Der Controller ruft die vorgesehene Funktion auf oder gibt eine

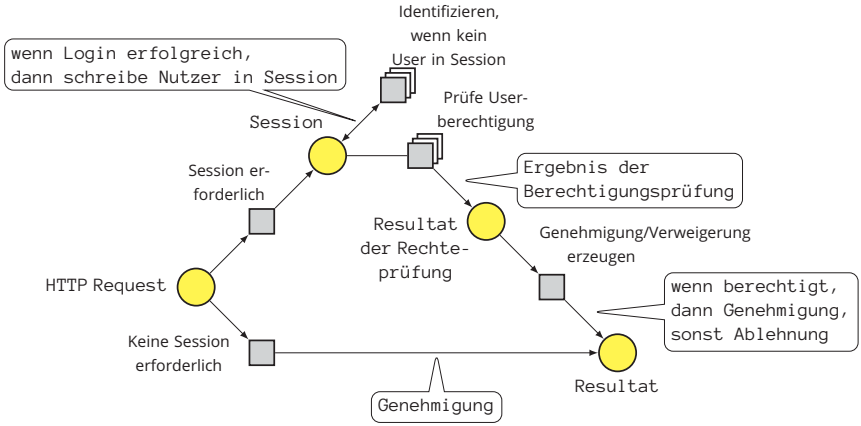


Abbildung 5.2: Entwurfsdetails der Rechteprüfung durch den Controller. Die zur Detaillierung gekennzeichneten Aktivitäten *Identifizieren, wenn kein User in Session* und *Prüfe Userberechtigung* werden in Abschnitt 5.1.3 verfeinert.

Fehlermeldung, die über die mangelnden Rechte informiert, aus. Sowohl Interaktion mit der Benutzerschnittstelle als auch Aufrufe der angebotenen Services werden auf diese Weise behandelt.

Implementierung Für die Implementierung werden von der jeweiligen XQuery-Implementierung abhängige Funktionen benötigt. So muss der Controller die URL analysieren sowie auf die Parameter des Requests und die Attribute der Session zugreifen, um zu entscheiden, wie mit der HTTP-Anfrage umzugehen ist. Anhand der URL und der darin übergebenen Parameter wird die angeforderte Ressource identifiziert, anhand der in der Session gespeicherten Informationen wird der die Anfrage stellende Nutzer identifiziert. Dazu muss die XQuery-Implementierung Funktionen zum Zugriff auf den HTTP-Request und ein Sessionmanagement bereitstellen. Die als XQuery Application Server geeigneten Implementierungen bieten hierzu eigene Bibliotheken und Funktionen an. Das DBMS eXist-db stellt Variablen im Namensraum `exist` für Zugriffe durch den Controller und Bibliotheken mit den Präfixen `request` und `session` zur Verfügung. Weitere Funktionen, auf die in den folgenden Implementierungen zugegriffen wird, werden in eXist-db durch die Bibliothek `util` bereitgestellt. Außerdem vereinfacht die Verfügbarkeit eines per XQuery zugreifbaren Mechanismus zum URL-Rewriting flexibles Routing durch Weiterleitung (`forward`)

und Umleitung (`redirect`) der Anfrage. Eine Weiterleitung geschieht Serverintern, während bei Umleitungen der Browser zum Absetzen einer neuen Anfrage aufgefordert wird. eXist-db implementiert dies dadurch, dass Module mit dem Namen `controller.xq*` per Konvention als Controller verwendet werden und deren Rückgabewert zum Festlegen des Routings ausgewertet wird [Ret12]. Dazu wird die Routinginformation als XML-Element `dispatch` zurückgegeben. Der Quellcode mit Beschreibung befindet sich in Abschnitt A.3.1.

View

Analyse Durch den Controller wird bestimmt, welche View-Komponente für die Ausgabe zuständig ist. Diese enthält ein Template zur Darstellung der Seite und ruft an den benötigten Stellen entsprechende Komponenten des Models auf, um auf die Geschäftslogik zuzugreifen.

Entwurf Ein View-Template wird als XQuery-Hauptmodul implementiert und gibt die erzeugte HTML-Seite zurück. Aufgrund der XML-Eigenschaften des hier verwendeten XHTML-Standards können deren Elemente mit XQuery gut bearbeitet werden. Innerhalb des Templates werden mit XQuery Zugriffe auf das Model implementiert. Falls eine Seite ein mittels XForms realisiertes Formular enthält, so wird das Formular im Ganzen eingebunden. Für ein Formular mit dynamisch erzeugtem Datenmodell wird dazu die entsprechende Transformation ausgeführt, für solche mit statischem Datenmodell kann die Implementierung direkt erfolgen.

Für von mehreren Views benötigte Elemente, wie z. B. Überschriften und Fußzeilen, werden Templates in Funktionen auslagert.

Implementierung Die Komponente generiert eine Webseite als XML-Dokument und greift dazu auf statisch und dynamisch erzeugte Inhalte zurück, die als Elemente an den passenden Stellen eingefügt werden. Statisch werden bspw. HTML-Metainformationen und Seitenbestandteile wie die Fußzeile eingebunden, dynamisch wird der aus dem Model bezogene eigentliche Inhalt der Seite generiert. Fragmente der Seite, die stets eine gleiche Struktur, aber einen anderen Inhalt aufweisen, wie der Seitentitel oder Fehlermeldungen, werden durch Funktionen, die die dynamischen Inhalte per Übergabeparameter erhalten, erzeugt. Der Quellcode mit Beschreibung befindet sich in Abschnitt A.3.1.

Model

Analyse Die Komponenten des Models werden von der Viewkomponente der jeweiligen Seite verwendet, um die zugehörigen Aufgaben auszuführen. Sie erhalten die dazu nötigen Parameter, führen die Geschäftslogik aus und geben ein Resultat zur Verwendung in der View zurück.

Entwurf Das Model wird durch XQuery-Bibliotheksmodule gebildet und stellt Funktionen bereit, die von den View-Komponenten verwendet werden. Diese kapseln die Geschäftslogik der Anwendung.

Implementierung Die Implementierung hängt von der Aufgabe der Funktion ab. Sie muss die in der View verwendeten Funktionsaufrufe implementieren und somit Funktionen mit entsprechender Signatur bereistellen. Ein Codebeispiel mit Beschreibung findet sich in Abschnitt A.3.1.

5.1.2 Datenmodell Nutzerverwaltung

Analyse Jeder individuelle Anwendungsbenutzer erhält eine Identität in der Webanwendung und zugehörige Identifizierungsmerkmale, also einen Nutzernamen und ein Passwort. Unterschiedliche Berechtigungsstufen der Webanwendung, Rollen, werden als Gruppen angelegt. Durch Bildung einer Hierarchie kann ausgedrückt werden, dass eine Gruppe die Rechte von anderen Gruppen umfasst. Im Unterschied zur rollenbasierten Zugriffssteuerung, bei der das Innehaben einer Rolle durch einen Nutzer bereits die gesamte Berechtigung ausdrückt, ergibt sich bei der hier verwendeten attributbasierten Zugriffssteuerung die Berechtigung durch die Zuordnung des Nutzer zu einer Gruppe unter Angabe der dazu benötigten Attribute. Wenn bspw. die aus Abschnitt 4.3.3 bekannten fachlichen Rollen Systemadministrator, Studienadministrator, Standortadministrator, Standortauswerter und Datensammler auf Gruppen abgebildet werden sollen, dann ist nur die Rolle Systemadministrator ohne Attribute zugeordnet, da sie global gilt. Für die Rolle Studienadministrator muss ein Attribut, das die Studie bezeichnet hinzugenommen werden und für die übrigen Rollen müssen Studie und Standort mit Attributen angegeben werden. Die in Abbildung 5.3 gezeigte Analyse erlaubt generisch die Zuordnung beliebig vieler Attribute zur Gruppenangehörigkeit eines Nutzers.

Entwurf Aus der gezeigten Analyse entsteht der in Abbildung 5.4 abgebildete Entwurf eines XML Schemas zur Speicherung der Nutzerdaten und Grup-

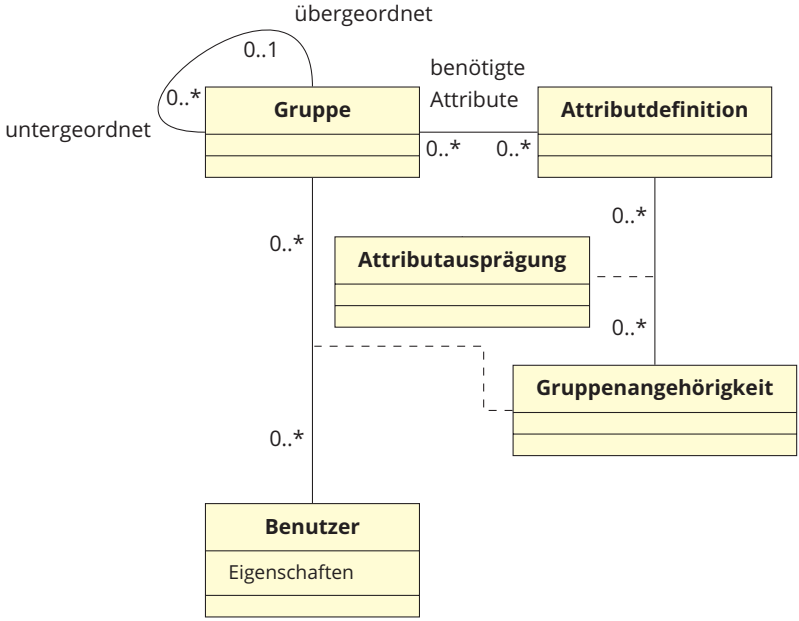


Abbildung 5.3: Analysemodell für die Datenstruktur der Benutzerverwaltung der hier vorgestellten XML-basierten Architektur.

pen. Ein group genanntes Element für eine Gruppe erhält die Attribute name als Schlüssel zur Identifizierung und ein Attribut displayName zur Anzeige. Es enthält außerdem beliebig viele Elemente `requiredAttribute`, die die Namen der von der Gruppe benötigten Attribute angeben. Gruppen sind als Hierarchie schachtelbar. Die Benutzer werden als Element user angelegt. Die Attribute name, password, salt und method dienen der Identifizierung, wobei Passwörter mit einem Salt versehen und nach der in method angegebenen Methode gehasht gespeichert werden [MT79]. Das Attribut activated gibt an, ob sich ein Nutzer einloggen kann, oder ob sein Zugang gesperrt ist. Jedes Element `user` kann beliebig viele Elemente `group` enthalten, die die Gruppenzugehörigkeiten angeben. Dabei muss die Gruppenzugehörigkeit als Fremdschlüssel für das Element `groupDefinition` dienen, damit nur tatsächlich vorhandene Gruppen referenziert werden. Jeder Gruppenzugehörigkeit sind `attribute`-Elemente zugeord-

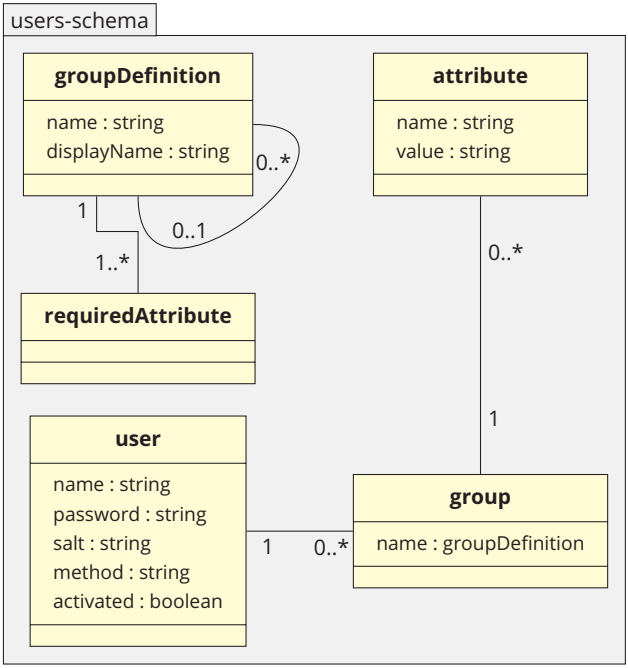


Abbildung 5.4: Entwurf für das XML Schema der Benutzerverwaltung der hier vorgestellten XML-basierten Architektur.

net, die den Attributnamen und dessen Ausprägung für den Benutzer enthalten.

Die Umsetzung als XML Schema `users.xsd` wird in Programm A.14 in Abschnitt A.3.2 gezeigt und beschrieben.

Implementierung Ein das XML Schema implementierendes Beispieldokument befindet sich in Programm A.15 in Abschnitt A.3.2.

5.1.3 Rechtemanagement

Das Rechtemanagement hat das Ziel, nur solche Zugriffe auf die Anwendung zuzulassen, die aus fachlicher Sicht berechtigt sind. Damit diese Prüfung nicht über die Module der Webanwendung verteilt implementiert werden muss, wird sie zentral durch den Controller gesteuert. Dazu gehören einerseits die Funktionen

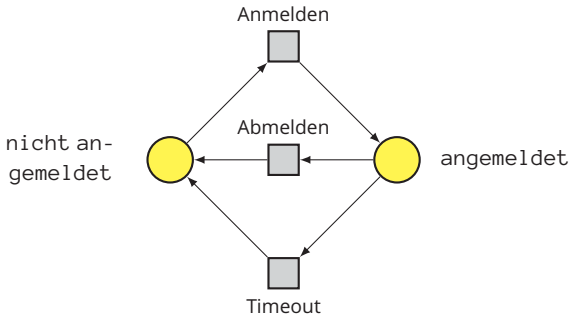


Abbildung 5.5: Analyse der Nutzeridentifizierung.

zur Identifizierung des Nutzers und andererseits die Autorisierung der Zugriffe. Die dazu vom Controller verwendeten Bibliotheken `app.auth` und `app.abac` werden im Folgenden vorgestellt. Grundsätzlich bieten DBMS eigene Lösungen zum Nutzermanagement, die den Zugriff regulieren. Diese basieren meist auf einem rein nutzer- oder rollenbasierten Modell und eignen sich nicht für die feingranularere, attributbasierte Zugriffssteuerung. Daher muss über dieses auf Ebene der Persistenzschicht agierende Nutzermanagement ein separates Nutzermanagement, das die Zugriffe auf die Geschäftslogik kontrolliert, eingefügt werden.

Analyse *Authentifizierung:* Das Anmelden des Nutzers geschieht durch Identifizierung mit Nutzernamen und Passwort, die in der Anfrage mitgesendet werden. Das Ausloggen geschieht entweder durch explizites Absenden einer Logout-Anfrage oder durch das Erreichen eines Timeouts, wenn für eine bestimmte Zeit Inaktivität festgestellt wurde, siehe Abbildung 5.5.

Autorisierung: Die Prüfung, ob ein Nutzer ausreichende Rechte für den angeforderten Zugriff hat, wird attributbasiert nach dem Prinzip *Attribute Based Access Control (ABAC)* vorgenommen. Dazu werden die Attribute des anfragenden Nutzers mit einem hinterlegten Regelwerk verglichen. Die Analyse dieser Aktivität ist bereits zur Einführung der Modellierungstechnik in Abbildung 3.5 auf Seite 91 gezeigt.

Entwurf *Authentifizierung:* Die Webanwendung selbst meldet sich zu Beginn einer Session mit einer eigenen DBMS-Nutzerkennung, die Zugriffsrechte für

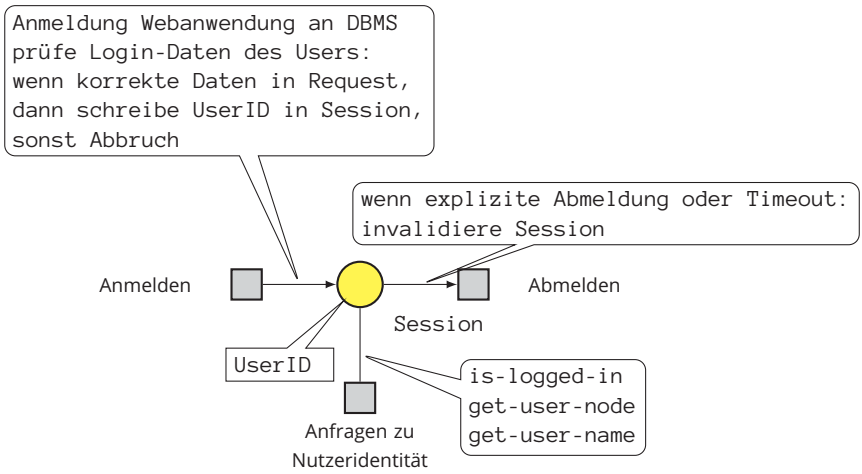


Abbildung 5.6: Entwurf der Nutzeridentifizierung.

alle von der Webanwendung benötigten Funktionen aufweist, am DBMS an. Danach werden die übermittelten Anmeldedaten des Benutzers geprüft. Bei korrekten Daten wird die Identität des angemeldeten Nutzers in die Session geschrieben, um sie im weiteren Verlauf der Interaktion nutzen zu können. Falsche Daten führen zu einem Abbruch der Identifizierung und dem Invalidieren der Session. Während der Interaktion mit dem Benutzer können andere Komponenten der Anwendung Anfragen zu dessen Identität stellen. Es kann geprüft werden, ob überhaupt ein Benutzer eingeloggt ist, wie sein Nutzername lautet und welche sonstigen Eigenschaften über ihn gespeichert sind, dazu zählen auch die Gruppenangehörigkeiten und die damit verbundenen Attribute. Nach Erreichen eines Timeouts oder expliziter Abmeldung wird die Session invalidiert. Abbildung 5.6 zeigt den Entwurf.

Autorisierung: Für jeden Request des Nutzers wird eine attributbasierte Rechteprüfung vorgenommen. Diese Aktivität ist ebenfalls bereits zur Einführung der Modellierungstechnik in Abbildung 3.7 auf Seite 93 gezeigt. Die darin generisch als `rules.xsd` und `entities.xsd` bezeichneten XML Schema für das Regelwerk und die zugreifenden Entitäten werden im Folgenden spezifischer als `abac.xsd` und `users.xsd` bezeichnet, um zu verdeutlichen, dass es sich um eine attributbasierte Rechtevergabe für den Zugriff durch Benutzer handelt. Die Datenstruktur des Rechtemanagements ist in Abbildung 4.10 auf Seite 144

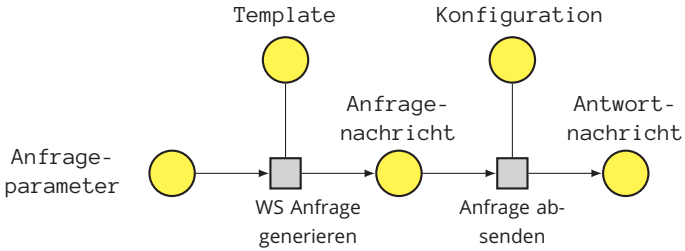


Abbildung 5.7: Analyse der Webservicenutzung.

gezeigt und im zugehörigen Abschnitt erläutert.

Implementierung Die Implementierung der Komponenten und die Beschreibung befinden sich in Abschnitt A.3.3.

5.1.4 Webservices

Im Folgenden wird eine Funktion zur Einbindung von Webservices entwickelt. Es wird davon ausgegangen, dass die zu verwendenden Webservices und deren Interfaces a priori bekannt sind.

Analyse Funktionen, die von der Webanwendung verwendet werden sollen, aber nicht durch Webanwendung implementiert werden, werden per Webservice fremdbezogen. Die Webanwendung fungiert dabei selbst als Client, der einen Request an den Webservice-Anbieter stellt und eine Antwort erhält. Dazu wird die Anfrage basierend auf den Anfrageparametern und einem Template generiert. Die erzeugte Nachricht wird an den durch die Konfiguration bestimmten Webserver gesendet und die erhaltene Antwort steht zur Weiterverarbeitung bereit. Abbildung 5.7 zeigt den Prozess.

Entwurf Die Webanwendung nutzt die seitens des Application Servers bereitgestellten HTTP-Client-Funktionen. eXist-db stellt diese im Modul `httpclient` bereit. Generell ist beim Entwurf festzustellen, ob der einzubindende Webservice SOAP- oder REST-basiert arbeitet. Der hier vorgestellte Entwurf ist für SOAP geeignet, da dies in der Systemumgebung des Anwendungsbeispiels x4T zum Einsatz kommt. Abbildung 5.8 zeigt den Entwurf. Die aus Sicht der Analyse atomar stattfindende Erzeugung der Anfrage ist im Entwurf in die Erzeugung

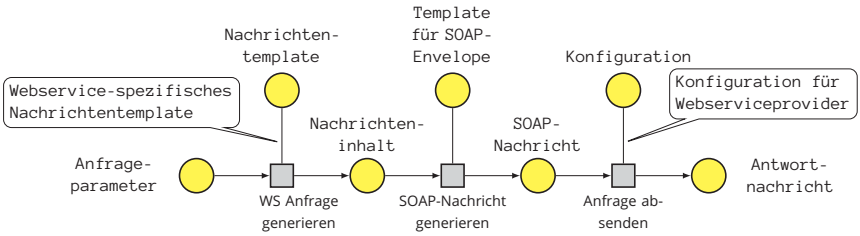


Abbildung 5.8: Entwurf für die Nutzung von SOAP-basierten Webservices.

des eigentlichen Nachrichteninhalts und in Erzeugung des SOAP-Envelopes, also des seitens des SOAP-Protokolls benötigten Nachrichtenformats, aufgeteilt.

Implementierung Der Quellcode zur Kommunikation mit Webservices und die dazugehörige Beschreibung befinden sich in Abschnitt A.3.4.

5.1.5 Testausführung

Die Eigenschaften von XQuery ermöglichen ein einfaches Testen der Module, wenn diese rein funktional implementiert sind. Da die Ausführung rein funktionaler Programme nicht von Programmzuständen abhängt, können solche Funktionen einfach durch den Aufruf mit festgelegten Parametern getestet werden, indem der berechnete Rückgabewert mit einem vorgegeben, zu erwartenden Wert verglichen wird. Falls, wie für viele Funktionen einer Webanwendung zu erwarten, keine rein funktionale Implementierung vorliegt, weil bei der Ausführung Daten gelesen oder gespeichert werden, muss vor den Tests solcher Funktionen ein bekannter Systemzustand hergestellt werden. Auf dieser Grundlage werden dann die einzelnen Testfälle ausgeführt und ausgewertet, so dass das Resultat produziert wird. Dabei wird nicht zwangsläufig nur der Rückgabewert getestet. Es können auch Systemzustände und andere Eigenschaften wie z. B. die Dauer der Ausführung ausgewertet werden. Vor Durchführung eines Testfalls muss sichergestellt sein, dass sich das System im korrekten Ausgangszustand für den Test befindet. Je nach Zeitbedarf für Herstellen des Ausgangszustandes ist abzuwägen, ob dieser nach jedem Test wieder herzustellen ist, oder ob Testfälle zu Klassen zusammengefasst werden, die hintereinander in derselben Umgebung ausgeführt werden. Bspw. könnte ein Test ein Objekt anlegen und der nächste könnte es löschen. Einerseits entsteht so eine Abhängigkeit zwischen

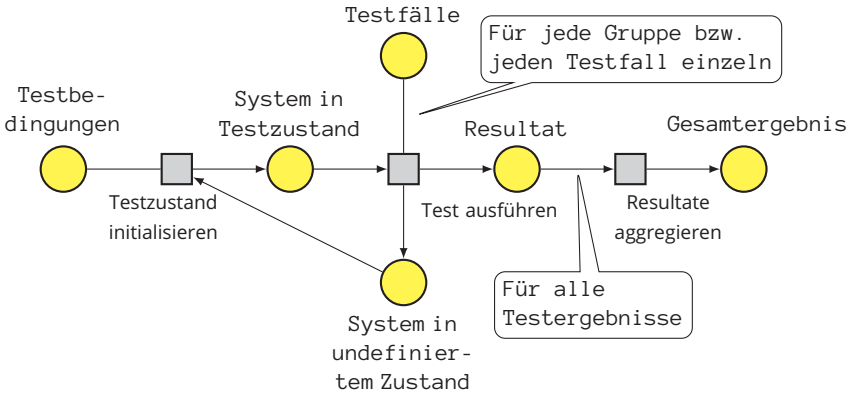


Abbildung 5.9: Ausführen der Testfälle.

den einzelnen Tests: Bei Fehlschlagen eines Tests verlieren die danach ausgeführten Tests ihre Aussagekraft. Anderseits sollte der Fehler, der zum Fehlschlagen eines Tests führt, sobald wie möglich repariert werden, sodass diese Abhängigkeit keine gravierenden Auswirkungen haben sollte. Vorteilhaft ist beim direkten Hintereinanderausführen von Tests eine geringere Testdauer. Dies fördert die Akzeptanz von Tests und ermöglicht ein häufigeres Ausführen. Nach Durchführung aller Tests werden die Resultate aggregiert und zum Gesamtergebnis des Testlaufs zusammengefasst. Abbildung 5.9 zeigt die Ausführung.

Dieses Vorgehen ist sowohl für Modultests als auch für Integrationstest anwendbar. Diese Tests können mittels der XQuery-Laufzeitumgebung direkt im Application Server ausgeführt werden und stehen somit jederzeit zur Verfügung. Bei manueller Ausführung der Tests können die Testergebnisse als HTML-Seite ausgeliefert und betrachtet werden, bei Ausführung durch die CI-Umgebung sind sie archivierbar.

Die hier gezeigte Implementierung ist angelehnt an [Ful08], da derzeit verfügbare XQuery Test-Frameworks (XQUT, xquery-unit) keine aktive Pflege erfahren. Im Gegensatz zu dem eXist-db-spezifischen und auf Annotationen basierendem XQSuite für Modultests ist sie flexibler und auch für Integrationstests verwendbar, bei denen neben dem Rückgabewert noch Systemzustände geprüft werden müssen. In Abschnitt A.3.5 wird die Implementierung des Frameworks beschrieben. Mit dem gezeigten Programmcode werden Modul- und Integrationstests durchgeführt.

Die Testfälle der Systemtests sind nicht XQuery-spezifisch. Hiezu bietet sich bspw. das Framework Selenium an. Der grundlegende Ablauf ist wie in Abbildung 5.9 gezeigt. Im Gegensatz zu den Modul- und Integrationstests steuern aber nicht die gezeigten XQuery-Funktionen den Testablauf, sondern das ausgewählte Framework.

5.1.6 Zwischenfazit

Während bei der Verwendung der traditionellen Architektur eine große Auswahl an fertigen Standardkomponenten in Form von umfassenden Frameworks und Bibliotheken zur Verfügung steht, fehlen diese bei Verwendung von XML-basierten Komponenten vielfach noch, so dass es eigener Ansätze bedarf. Für die Entwicklung von Webanwendungen häufig benötigte Komponenten wurden in diesem Abschnitt in XQuery entworfen und implementiert und stehen zur Nutzung im Rahmen der hier vorgestellten XML-basierten Architektur zur Verfügung. Weitere ggf. benötigte Komponenten können nach demselben Prinzip entwickelt werden.

5.2 Übertragbarkeit in Echtanwendung am Beispiel x4T

5.2.1 Vorstellung Single-Source und x4T

Universitätskliniken, wie das UKM[1], dienen neben der medizinischen Behandlung der Patienten auch Forschung und Lehre. In der medizinischen Forschung spielen klinische Studien ein wichtige Rolle, bei denen Erkenntnis aus der Beobachtung von Probanden unter kontrollierten Bedingungen gewonnen wird [SS08; MC05]. Diese Beobachtungen werden vom Studienpersonal in *Case Report Forms (CRFs)* dokumentiert. Klassischerweise wird die Dokumentation der Studie redundant zu der Behandlungsdokumentation erhoben und aufgezeichnet, was als *Dual-Source* bezeichnet wird. Die Behandlungsdokumentation wird dabei in einem *KIS* gespeichert. Die Studiendokumentation erfolgt wie im Studienprotokoll festgelegt, die CRFs können sowohl papierbasiert als auch digital sein.

Bereits ohne die Durchführung klinischer Studien verbringen Ärzte 20 %–27 % ihrer Arbeitszeit mit der Dokumentation ihrer Tätigkeit [AS09; TFO+10]. Eine zusätzliche Belastung durch die Ausweitung der Dokumentation bei Studiendurchführung ist daher nicht wünschenswert. Daher werden in der Literatur Ansätze zur Wiederverwendung der Behandlungsdokumentation im Rahmen von „Secondary Use“ zu Studien- oder sonstigen Zwecken diskutiert [SBH+07; PG09].

In der Literatur sind Implementierungen beschrieben, die jeweils eine hohe Abhängigkeit zum verwendeten KIS aufweisen oder als Data-Warehouse (DWH) nicht in Echtzeit Daten bereitstellen. So beschreibt [KAR+07] einen Prototypen, bei dem Daten zunächst in gesonderten Formularen erfasst werden und danach im KIS als Behandlungsdokumentation und im CRF als Studiendokumentation gespeichert werden. In [BSM+11] ist eine Studie beschrieben, bei der das Format ODM zur Spezifikation der Formulare verwendet wird, die Implementierung aber mit KIS-spezifischen Mitteln erfolgt. [FRL+11] beschreibt ein System, bei dem die Datenintegration durch das KIS vorgenommen wird, was eine entsprechende Funktionalität voraussetzt. In [KKS+13] wird eine Studie beschrieben, bei der das Vorbelegen der CRFs mit Daten aus dem KIS erfolgt. Es wurde gezeigt, dass das Ausfüllen vorbelegter CRFs den Prozess beschleunigt. Allerdings kamen hierbei papierbasierte CRFs zum Einsatz, die mit entsprechenden Daten bedruckt wurden, und keine elektronische Datenerfassung.

Ein Konzept für ein klinikweites DWH ist in [WJR12] vorgestellt. Da ein

[1] http://www.klinikum.uni-muenster.de/

DWH als analytisches System klassischerweise keine Daten für den operativen Betrieb bereithält, liegen diese nicht in Echtzeit vor, sondern erst nach Durchlaufen eines möglicherweise nur periodisch stattfindenden *Extrahieren, Transformation, Laden (ETL)*-Prozesses. Für das vorliegende Single-Source Szenario ist diese Lösung nicht befriedigend, da es sich bei der Integration von Daten aus der Behandlungsdokumentation in die Studiendokumentation um eine operative Tätigkeit handelt. Um für das Studienpersonal eine konsistente Sicht zu erhalten, muss mit Daten aus dem operativen KIS gearbeitet werden, weswegen der Ansatz eines DWH keine Berücksichtigung findet. Ein Beispiel für ein standortübergreifendes klinisches DWH ist in [MBK+11] beschrieben.

Der im Rahmen des Single-Source-Projekts verfolgte Ansatz sieht eine Single-Source-Architektur x4T [DFB+11] vor, die weitestmöglich unabhängig von der Implementierung des KIS arbeitet sowie studien- und standortübergreifend einsetzbar ist. Eine ins Detail gehende Darstellung des x4T-Single-Source-Prozesses, der Architektur und ausgewählter Anwendungsfälle ist in [FBL+14] zu finden. Um die im KIS erfassten Daten, die für eine Studie benötigt werden, nicht redundant zu erfassen, werden sie zur Vorbelegung der entsprechenden Felder der CRFs verwendet. Um die medizinische Bedeutung der Datenelemente maschinenlesbar anzugeben, werden sie mit semantischen Annotationen versehen. Dazu werden die verwendete medizinische Nomenklatur und die innerhalb dieser verwendeten Begriffe angegeben. Das x4T-Konzept unterstützt beliebige Begriffssysteme. Begriffe werden durch Angabe eines Identifikators für das System und eines oder mehrerer innerhalb dessen gültiger Ausdrücke spezifiziert. So können z. B. *SNOMED CT*[2] [Don06] oder *LOINC*[3] [FMD+96] verwendet werden. Durch die Angabe `SNOMED` und des Codes `184099003` wird ein Wert bspw. mit der Bedeutung *Geburtsdatum* versehen. Bei übereinstimmender Bedeutung eines Datums im KIS und einer Fragestellung auf dem CRF werden die Daten aus dem KIS integriert.

Dazu wird für das jeweilige KIS ein Adapter implementiert, der den Zugriff auf die klinischen Daten vereinheitlicht. Die digitalen CRFs werden durch die Komponente x4T-Formularmanagement, die mittels der in dieser Arbeit beschriebenen XML-basierten Architektur für Webanwendungen implementiert ist, verwaltet. Der Zugriff auf die Webanwendung ist aus der Behandlungsdokumentation des Patienten im KIS möglich. Das Studienpersonal kann darin die vorbelegten CRFs einsehen und ergänzen. Im Rahmen einer Studie wird ein Patient unter einem Pseudonym geführt, dessen Depseudonymisierung nur in-

[2] http://www.ihtsdo.org/snomed-ct/

[3] https://loinc.org/

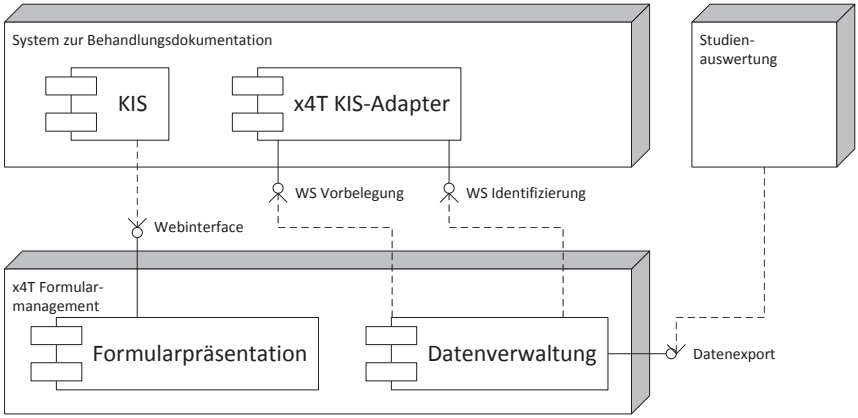


Abbildung 5.10: Übersicht über die x4T-Architektur. Die Komponente x4T-Formularmanagement ist als XML-basierte Architektur realisiert, wie in dieser Arbeit beschrieben.

nerhalb des KIS stattfinden kann. Die im Rahmen einer Studie verarbeiteten Daten dürfen aus Gründen des Datenschutzes keine Rückidentifizierung erlauben. Dieses Pseudonym wird verwendet, um die zu dem Studienteilnehmer passenden Vorbelegungsdaten über den KIS-Adapter anzufragen. Der ebenfalls durch diesen Adapter bereitgestellte Identifizierungsdienst erlaubt die Identifizierung von KIS-Benutzern im Sinne eines Single Sign-on (SSO). Eine Schnittstelle für den Datenexport erlaubt den Transfer der erhobenen Studiendaten in weitere Systeme, z. B. zur Auswertung und Archivierung. Abbildung 5.10 zeigt eine Übersicht der Komponenten.

5.2.2 Analyse

Während der Analyse wurden anhand von drei am UKM durchgeführten Studien aus den Bereichen Onkologie, Dermatologie und Gefäßchirurgie durch Interviews mit dem Studienpersonal die Anforderungen an ein Single-Source-System ermittelt. Diese Anforderungen wurden von den jeweiligen Einzelfällen abstrahiert und mit dem Ziel eines studienübergreifend verwendbaren Konzepts weiterentwickelt. Daraus ergeben sich folgende funktionale Anforderungen [BFD12]:

- Funktionen zur Studiendurchführung

- Export in ein Format zur weiteren Verarbeitung
- Einfache deskriptive Statistikfunktionen
- Einfache Plausibilitätsprüfungen auf CRFs
- Aktualisierung der CRFs
- Vorbelegungsmöglichkeit für CRFs
- Automatische Übernahme der Behandlungsdokumentation in CRFs

Die Funktionen zur Studiendurchführung dienen dem Anlegen neuer Studien, dem Verwalten von Benutzern, dem Hinzufügen von Studienteilnehmern und dem Ausfüllen der Formulare.

Datenexporte sind im CDISC ODM-Format oder im generischen Comma-separated Values (CSV)-Format vorgesehen. Das ODM-Format ermöglicht einen Export von Daten und Metadaten und eine Weiterverarbeitung mit allen kompatiblen Programmen aus dem Studienkontext. Es vermeidet einen Lock-in-Effekt und gewährleistet aus technischer Sicht die freie Weiterverwendung der Daten. Das CSV-Format wird von Statistik-Software zur Auswertung benötigt.

Die deskriptiven Statistiken geben Hinweise zum Stand der Studiendurchführung, bspw. über die Anzahl der Studienteilnehmer und deren Verteilung hinsichtlich bestimmter Merkmale.

Plausibilitätsprüfungen schränken den Wertebereich der gültigen Antwortmöglichkeiten ein, bspw. werden bei der Frage nach einem Datum nur existierende Daten zugelassen oder der Wertebereich eines numerischen Wertes wird sinnvoll eingeschränkt. Eine weitere Möglichkeit, die Plausibilität der Antworten zu erhöhen, ist das Erheben von Daten in Abhängigkeit zu vorherigen Antworten. So ist z. B. die Frage nach einer Schwangerschaft nur bei Teilnehmern weiblichen Geschlechts sinnvoll.

Falls während des Studienverlaufs aus fachlicher Sicht Änderungen an den CRFs vorgenommen werden müssen, sollten sich diese möglichst einfach in eine bereits laufende Studie integrieren lassen.

Die Punkte Vorbelegung und automatische Übernahme betreffen den Kern des Single-Source-Konzeptes. Per Vorbelegung werden Daten aus dem KIS extrahiert und in einen logisch und visuell vom Datenbereich unterschiedenen Vorbelegungsbereich eines CRFs kopiert. Bei Bearbeitung eines CRF werden sie explizit vom Studienpersonal kontrolliert und in den Bereich der Studiendaten übernommen oder es werden anderweitig erhobene Werte dokumentiert. Diese Methode bietet sich an, wenn die Datenqualität im KIS nicht überprüft wird oder

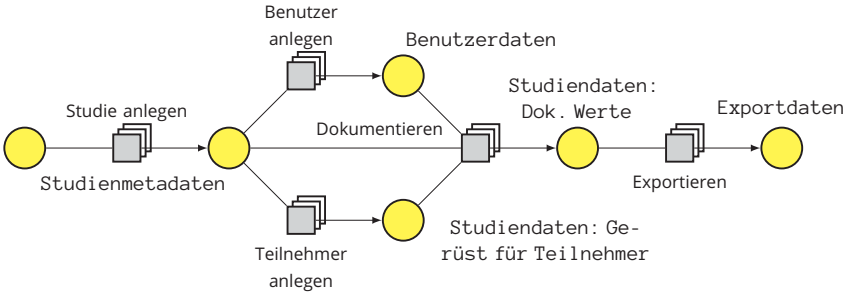


Abbildung 5.11: Abstrakte Sicht auf den Prozess der Studiendurchführung.

dort mehrere Ausprägungen eines Datums vorliegen und nicht ex ante anhand der Metadaten zu entscheiden ist, welche dieser Ausprägungen in die Studiendokumentation übernommen werden soll.

Falls die Datenqualität bereits im KIS geprüft wird und eine eindeutige Zuordnung eines CRF-Feldes zu einem Datum im KIS existiert, kann die automatische Dokumentation verwendet werden. Dabei werden Daten aus dem Vorbelegungsbereich ohne manuelle Kontrolle übernommen. Der Status der Werte ist dann identisch mit dem von manuell durch das Studienpersonal eingetragenen Werten.

Abbildung 5.11 zeigt eine Sicht auf Ebene 0 des Prozesses der Studiendurchführung.

Als nicht-funktionale Anforderungen wurden identifiziert:

- Einfache Bereitstellung
- Schnelle Einrichtung neuer Studien
- Geringe Kosten für Inbetriebnahme und Einsatz
- Geringer Schulungsbedarf und gute Benutzbarkeit

Die Bereitstellung des Systems soll mit geringem Aufwand verbunden sein. Dazu sollte ein Zugriff auf die Studieninfrastruktur ohne Installation auf einzelnen Arbeitsplätzen möglich sein, da bei den häufig zentral verwalteten Arbeitsplätzen eines Klinikums die Installation von Software erst nach Durchlaufen einer Freigabeprozedur erfolgen kann.

Die Einrichtung des Systems für neue Studien soll schnell möglich sein, d. h. Programmierung der jeweils studienspezifischen Anforderungen soll zugunsten

von Konfiguration vermieden werden. Die Definition der Studien sollte daher in einem anerkannten Standardformat erfolgen, das von der Anwendung interpretiert werden kann.

Gerade für wissenschaftsinitiierte Studien, die meist ohne finanzstarken Sponsor stattfinden, sind geringe Kosten für IT-Unterstützung von Bedeutung. Für den Einsatz sollen keine Lizenzkosten anfallen. Ebenfalls sind extreme Anforderungen an die Laufzeitumgebung oder besonderer Wartungsbedarf zu vermeiden.

Schließlich werden – nicht spezifisch für die Studienunterstützung – eine intuitive Bedienbarkeit und gute Benutzbarkeit gefordert, so dass auf aufwändige Softwareschulungen verzichtet werden kann und die Bedienung sich möglichst gut in den Arbeitsablauf integriert.

5.2.3 Entwurf

Im Folgenden werden die in der Analyse beschriebenen Funktionen detailliert für die Implementierung vorbereitet.

Studie anlegen

Das Anlegen einer Studie umfasst das Prüfen und Speichern einer im Vorfeld angelegten ODM-Datei, die die Metadaten der Studie enthält. Zunächst muss geprüft werden, ob es sich bei der hochgeladenen Datei um eine wohlgeformte XML-Datei handelt. Falls dies zutrifft, wird die hochgeladene Datei validiert. Das dazu verwendete Schema ist ein ODM-Schema mit x4T-spezifischen Vendor Extensions. Vor dem Speichern der Datei im System muss noch sichergestellt werden, dass das Attribut `@OID` der Studiendefinition als Primärschlüssel verwendet werden kann und somit noch keine Studie mit demselben Attributwert im System existiert. Damit ist die hochgeladene ODM-Datei als Studie im System verfügbar. Abbildung 5.12 zeigt den Vorgang.

Benutzer anlegen

Zu einem Benutzer gehören die studienübergreifenden Anmeldedaten für das x4T-System und die studienspezifischen Daten des ODM-AdminData-Bereichs. Diese werden gemeinsam in einem Formular erfasst. Sodann wird geprüft, ob die gewählten Parameter valide sind, also insbesondere ob der angegebene Nutzername als ID eindeutig ist und ob der anlegende Nutzer über ausreichende Rechte

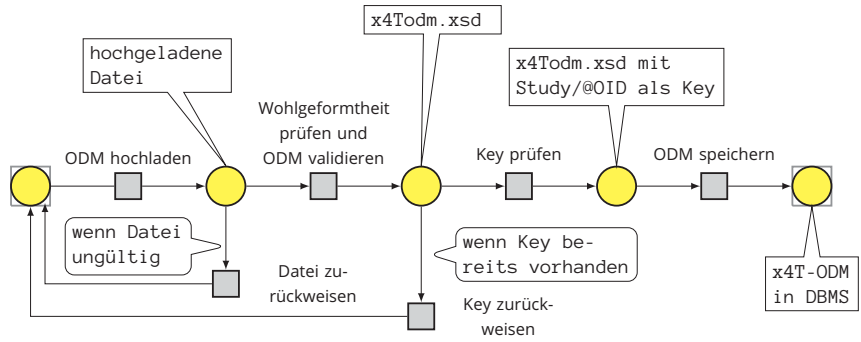


Abbildung 5.12: Verfeinerung der Aktivität *Studie anlegen*.

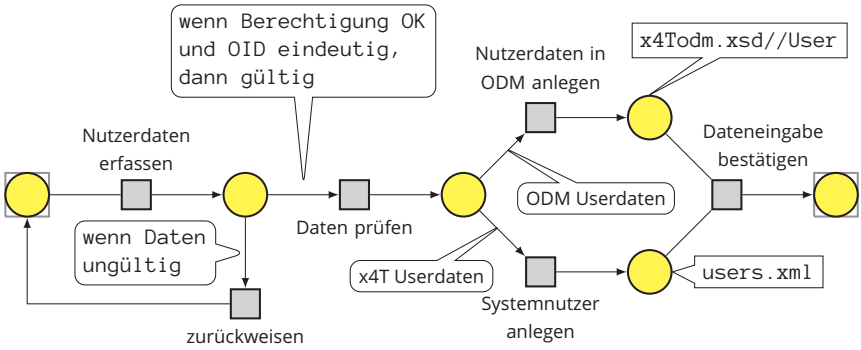


Abbildung 5.13: Verfeinerung der Aktivität *Benutzer anlegen*.

für die gewählten Parameter verfügt. So darf z. B. ein Benutzer, der Studienadministrator ist, nur Benutzer zu der zugehörigen Studie hinzufügen, nicht aber zu Studien, in denen er dieses Recht nicht hat. Schließlich werden die Daten in die jeweils zugehörigen Dokumente geschrieben. Der Prozess ist in Abbildung 5.13 abgebildet.

Teilnehmer anlegen

Nach dem Anlegen der x4T-Benutzer müssen noch die Teilnehmer einer Studie angelegt werden, bevor dokumentiert werden kann. Für das Anlegen von Teilnehmern stehen zwei Wege zur Verfügung: Das Anlegen über die Schnitt-

stelle zum KIS, bei dem eine Verbindung mit einer Patientenakte angelegt wird und das manuelle Anlegen über die Benutzeroberfläche ohne Verknüpfung mit einem KIS. Die erste Variante ist die im Single-Source-Einsatz verwendete und wird aus dem KIS heraus initiiert, die zweite wird aus der Webanwendung gestartet. Sie kann für das Anlegen von Studienteilnehmern verwendet werden, die nicht als Patienten im KIS vorhanden sind, oder falls eine Verknüpfung manuell hergestellt werden soll. Es müssen die jeweils zu dem Studienteilnehmer gehörenden Daten angegeben werden, also die Studie, die Version, der Benutzername der ausführenden Person und der Ort. Falls sich der Benutzer an der Webanwendung angemeldet hat, liegen diese Daten bereits vor, falls der Benutzer das Anlegen über das KIS auslöst, werden sie über den Webservice bezogen. Außerdem wird, falls das Anlegen aus dem KIS geschieht, ein temporärer Identifikator für den Teilnehmer übergeben. Falls das Anlegen aus der Webanwendung geschieht, kann manuell ein Pseudonym gewählt werden. Wenn kein manuell festgelegtes Pseudonym vorliegt, wird durch x4T ein Pseudonym erzeugt. Nachdem die Daten der Webanwendung bekannt sind, werden sie auf Vollständigkeit und Plausibilität geprüft und zur Erzeugung der ODM SubjectData-Struktur verwendet. Ein XSLT-Programm erhält sie als Parameter und generiert auf Basis der ODM-Studiendefinition diese Struktur. Diese wird im DBMS persistiert und steht zur Dokumentation der Daten des Teilnehmers zur Verfügung. Abschließend wird, falls der Aufruf durch das KIS initiiert wurde, diesem eine Bestätigung über das Anlegen des Studienteilnehmers geschickt, in der das generierte Pseudonym und der zugehörige temporäre Identifikator übertragen werden. Falls der Aufruf innerhalb der Webanwendung erfolgte, wird dem Benutzer das Anlegen des Teilnehmers am Bildschirm bestätigt. Abbildung 5.14 zeigt den Prozess zum Anlegen eines Studienteilnehmers.

Dokumentieren

Nach dem Anlegen der notwendigen Daten kann die Dokumentation der Studiendaten eines Teilnehmers durchgeführt werden. Diese besteht im Single-Source-Fall aus der Durchführung der Vorbelegung und der automatischen oder manuellen Übernahme der Daten sowie ggf. dem manuellen Ergänzen von Werten. Zur Durchführung der Vorbelegung wird aus den Studienmetadaten, insbesondere den semantischen Annotationen, und dem Pseudonym des Teilnehmers eine SOAP-Nachricht generiert, mit der das x4T Formularmanagement die benötigten Daten vom Webservice des KIS-Adapters anfordert. Die Nachricht wird mittels eines XSLT-Programms erzeugt. Die Antwort des Webservices enthält die angeforderten Daten, deren semantische Annotation und Metadaten, die ihre

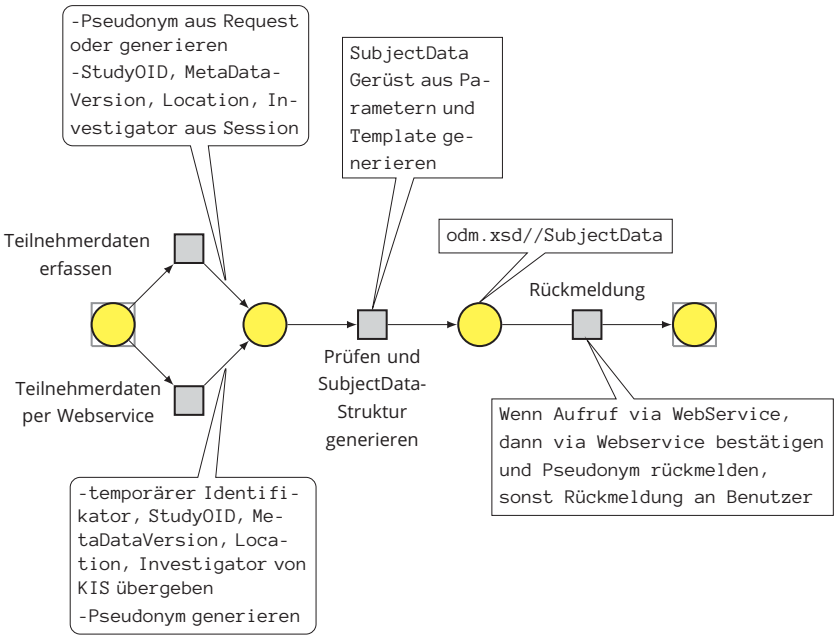


Abbildung 5.14: Verfeinerung der Aktivität *Teilnehmer anlegen*.

Herkunft im KIS spezifizieren: Einen die Formularinstanz eindeutig bestimmenden Schlüssel und als Metadaten zu dieser Instanz das Modifikationsdatum und den Namen des Formulars. Diese Daten werden im nächsten Schritt in den Vorbelegungsbereich der Studiendaten des Teilnehmers integriert. Dazu wird die in Abbildung 5.15 gezeigte Erweiterung des ODM-Schemas vorgenommen. Zu jedem Element ItemData des ODM-Standards kann ein Element prepopDataSet im Namensraum prepop angelegt werden. Zu diesem gehört mindestens ein Element codeData, das semantische Annotationen zur Bedeutung der enthaltenen medizinischen Daten enthält. Die referenzierte Terminologie wird als codeSystemOID angegeben, die Identifizierung des Konzepts innerhalb der Terminologie wird in code festgehalten. Das Element prepopDataItem dient zur Speicherung der über den KIS-Adapter erhaltenen Vorbelegungsdaten. Die hier als Attribute modellierten Eigenschaften formName, sourceInstanceID und entryDate enthalten Informationen zur Datenherkunft, die u. a. dem dokumentierenden Personal zur Einschätzung der Verwendbarkeit der Daten im Rahmen der Studie dienen. Die

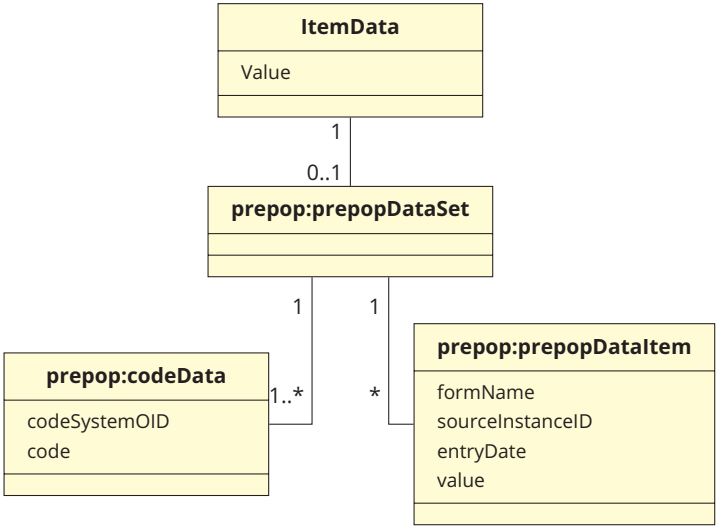


Abbildung 5.15: Erweiterung des ODM-Datenmodells als `x4Todm.xsd` zur Speicherung der Vorbelegungsdaten.

Eigenschaft value speichert den zur Vorbelegung vorgesehenen Wert.

Anhand der Metadaten wird das zugehörige Element identifiziert, anhand der Formularinstanz ist erkennbar, ob es sich um ein neues oder bereits vorhandenes Datum handelt. Für neue Daten wird ein neuer Eintrag angelegt, vorhandene Daten werden bei unterschiedlichem Modifikationsdatum aktualisiert. Im Fall der automatischen Dokumentation erfolgt danach ein Kopieren des Wertes aus dem Vorbelegungsbereich in den Dokumentationsbereich. Danach können die Daten im Dokumentationsformular angesehen und bearbeitet werden. Der Prozess ist in Abbildung 5.16 gezeigt. Da die Aktivität der Formularbearbeitung zentral für das x4T Formularmanagement ist und die hier präsentierte XML-basierte Architektur dabei besonders vorteilhaft ist, wird sie im Folgenden näher erläutert.

Das Dokumentationsformular wird mittels XForms realisiert und dynamisch zur Laufzeit aus den Metadaten der Studie und den Daten des Teilnehmers durch ein XSLT-Programm generiert. Hierbei kommen die Vorteile der XML-basierten Architektur besonders zum Tragen: Es müssen keine Transformationen der Daten zwischen Persistenzschicht, Geschäftslogik und Präsentationsschicht vorgenommen werden. Der zu dem Studienteilnehmer gehörende Teilbaum mit

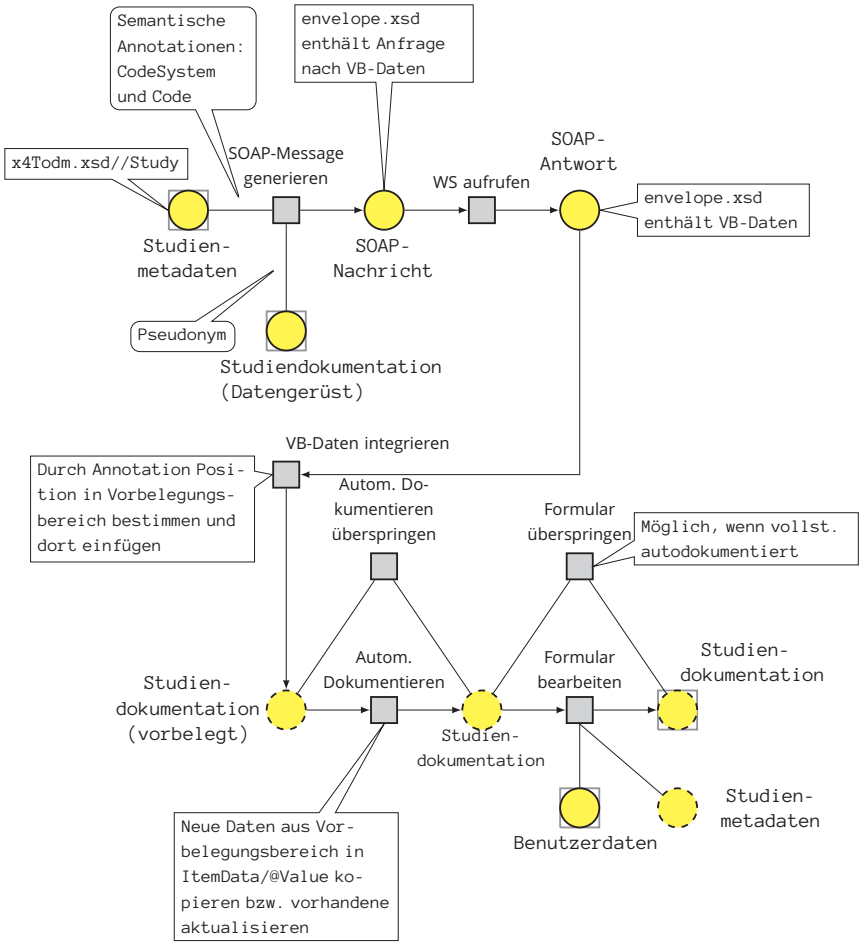


Abbildung 5.16: Verfeinerung der Aktivität *Dokumentieren*.

dem Wurzelelement `SubjectData` der in der Persistenzschicht vorhandenen ODM-Struktur wird durch die Geschäftslogik der Präsentationsschicht zugängig gemacht und dient innerhalb von XForms als Teil des Datenmodells. Die Darstellungskomponenten des Formulars und ggf. benötigte Bindungen an Datentypen und Restriktionen werden aus den Metadaten abgeleitet. Jede Modifikation durch den Benutzer ändert die in XForms vorliegenden ODM-Daten des Teilnehmers. Nach Abschluss der Bearbeitung wird der ODM-Teilbaum wieder an die Geschäftslogik übergeben, die die Speicherung in der Persistenzschicht der Webanwendung veranlasst. Auch dazu ist keine Transformation der Daten nötig.

Exportieren

Nach Abschluss der Dokumentation liegen die Daten des Teilnehmers vor und können zur Auswertung exportiert werden. Abbildung 5.17 zeigt den Vorgang. Dabei stehen drei Optionen zur Verfügung. Der Export als standardkonformes ODM-Dokument, der Export im CSV-Format als Rohdaten zur Analyse mittels Statistik-Software und der Export als Portable Document Format (PDF)-Dokument als menschenlesbares Schriftstück. Die Teilnehmerdaten werden vor dem Export gefiltert: Zunächst obligatorisch, um für den angemeldeten Benutzer nur zulässige Daten zu erhalten und dann optional, um dem Benutzer eine weitere Einschränkung der Teilnehmer zu erlauben.

Der ODM-Export der gefilterten Teilnehmerdaten ist einfach zu realisieren, da lediglich die hinzugefügten Elemente und Attribute in den Namensräumen der Vendor Extensions entfernt werden müssen.

Der CSV-Export gestaltet sich aufgrund wiederholbarer Elemente, die wie in Abschnitt 3.4.4 erläutert in ODM durch das Attribut `Repeating="YES"` ausgezeichnet werden, und durch die notwendige Transformation von einer Baum- in eine Tabellenstruktur etwas komplizierter: Zunächst muss ermittelt werden, welche Wiederholungen in allen Daten der zu exportierenden Teilnehmer maximal enthalten sind. Aus diesen bestimmen sich die Tabellenspalten und somit die Kopfzeile des CSV-Exports, die im nächsten Schritt konstruiert wird. Für jede Zeile wird eine Bezeichnung generiert, die sich aus der Hierarchie der Studiendefinition und einem Zähler für die sich wiederholenden Datenelemente zusammensetzt. Die Bezeichnung spiegelt die Baumstruktur der ODM-Daten wieder. Dies geschieht durch Konkatenation der jeweiligen ODM-OIDs und ODM-Repeatkeys mit einem Punkt als Trennzeichen nach dem Schema `StudyEventOID.StudyEventRepeatKey.FormOID.FormRepeatkey.ItemGroup.ItemGroupRepeatkey.ItemDataOID`, wobei

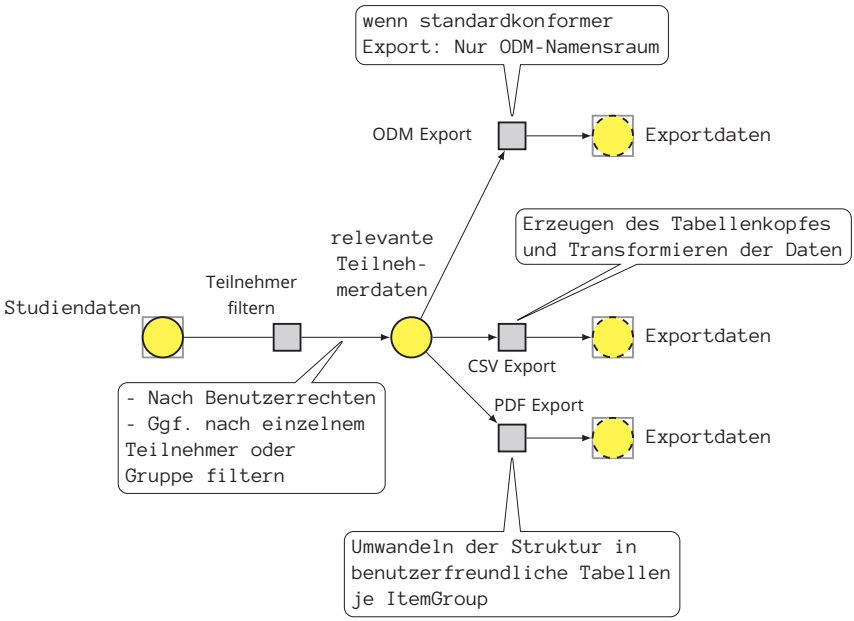


Abbildung 5.17: Verfeinerung der Aktivität *Exportieren*.

die Zeichenkette für Repeatkey leer bleibt, falls das dazugehörige Element nicht wiederholbar ist. Dann wird über alle zu exportierenden Teilnehmer iteriert und für jeden Teilnehmer eine Tabellenzeile angelegt. Dabei wird in jeder Tabellenspalte geprüft, ob für den Teilnehmer Daten für die in Frage kommende Zelle vorliegen und diese werden ggf. hineingeschrieben.

Der PDF-Export bildet die Formularstruktur auf ein seitenorientiertes Schriftstück ab und nutzt dazu die in Abschnitt 3.4.3 vorgestellte XSL FO-Technik. Dazu muss die hierarchische ODM-Darstellung in eine lineare Repräsentation transformiert werden. Die in der ODM-Hierarchie einer ItemGroup untergeordneten Daten werden in einer Tabelle zusammengefasst und unter Angabe des übergeordneten Formulars und des Studienereignisses im Zieldokument dargestellt. Für jedes Formular soll dabei eine neue Seite begonnen werden.

Auswertung

Die Auswertung der erhobenen Daten liegt nicht mehr im Anwendungsbereich von x4T. x4T bietet einfache deskriptive statistische Maße, wie die Anzahl der Patienten oder die Verteilung bestimmter Merkmalsausprägungen in den erhobenen Daten, um einen Überblick über den Stand der Studiendurchführung zu vermitteln. Für die Analyse der Daten hinsichtlich medizinischer Fragestellungen wird auf die exportierten Daten zurückgegriffen. Von diesen eignet sich das CSV-Format zur Analyse mittels gängiger Statistik-Software. Das CSV-Format enthält aber außer den Spaltenüberschriften keine Metadaten. Die zur Interpretation benötigten Metadaten müssen separat übermittelt werden. Die Standardisierung eines Formats, das die Studienmetadaten stärker als CSV repräsentiert und als Datenformat für Statistik-Software geeignet ist, ist noch nicht abgeschlossen [Bab13]. Exporte im ODM-Format enthalten die vollständigen Metadaten der Studie und können daher neben dem Zweck der Archivierung auch zur Erzeugung weiterer, ggf. später zur Auswertung benötigter Formate verwendet werden.

5.2.4 x4T Implementierung

Komponenten

Um die Software ohne Lizenzkosten oder die Beachtung restriktiver Verwendungsauflagen im klinischen Alltag einsetzen und vervielfältigen zu können, sollen die Komponenten so gewählt werden, dass sie unter einer freien Softwarelizenz stehen.

Von den in Abschnitt 4.4.1 angeführten XML-DBMS sind folgende unter einer freien Lizenz erhältlich: *BaseX*, *eXist-db* und *Sedna*. Die drei genannten Programme werden hinsichtlich ihres Funktionsumfangs, der Unterstützung für Anwender und der Entwickleraktivität gesichtet.

Der Funktionsumfang wird der Dokumentation entnommen und anhand der Software nachvollzogen. Insgesamt integriert eXist-db standardmäßig die meisten Funktionen, insbesondere unterscheidet sich die Software durch die Integration des serverseitigen XForms-Interpreters betterFORM, die Unterstützung von XSL FO (s. Abschnitt 3.4.3) und einen Scheduler zum asynchronen und zeitversetzten Aufruf von Programmcode von den beiden anderen. Aufgrund des Funktionsumfangs lässt sich eXist-db als Web-Application-Server [SKM00] mit integriertem DBMS auffassen. Da eXist-db die umfangreichste Grundausstattung enthält und die aktivste Entwicklung sowie Community aufweist, fällt die Wahl auf dieses DBMS.

Hinsichtlich der XForms-Funktionalität bieten die beiden frei erhältlichen serverseitigen Interpreter *betterFORM* und *Orbeon Forms* die benötigten Eigenschaften. Die Integration in eXist-db erleichtert das Installieren und Konfigurieren der Anwendung, weswegen für x4T auf betterFORM als XForms-Interpreter zurückgegriffen wird.

Infrastruktur und Testen

Um agil zu entwickeln, ist die automatische Ausführung von Testfällen besonders wichtig. Mittels eines Codeverwaltungssystems und eines CI-Systems kann jede in das Code-Repository eingepflegte Änderung ohne Zutun des Entwicklers getestet werden. Im Fehlerfall benachrichtigt der CI-Server den Entwickler, damit der fehlerhafte Code bereinigt werden kann.

Zur Entwicklung von x4T wird dazu auf das Codeverwaltungssystem SVN und den CI-Server Hudson zurückgegriffen. Bei jeder Übermittlung von Code in das Repository wird eine Benachrichtigung an den CI-Server ausgelöst. Hudson kopiert die aktuelle Version des Codes in eine Testumgebung, führt die Testfälle aus und archiviert das Ergebnis. Besteht die Version die Tests, wird eine Kopie in einem Bereich des Code-Repositorys angelegt, die stets den installationsfertigen Code enthält. Zum Durchführen von Modul- und Integrationstests wird das in Abschnitt 5.1.5 beschriebene, selbstentwickelte Gerüst verwendet. Für Systemtests wird das Framework Selenium verwendet. Damit erstellte Testfälle können ebenfalls durch den CI-Server ausgeführt werden.

Bei der Wahl der Programmierwerkzeuge hat sich der Editor Vim bewährt. Dafür ist eine XQuery Syntax-Hervorhebung verfügbar, die Sprachkonstrukte bei der Programmierung visuell kennzeichnet. Eine Unterstützung zur automatischen Formatierung von Code besteht ebenfalls. Da die Entwicklung der gesamten x4T-Software von einem kleinen Team aus drei Personen geleistet wurde, konnte auf eine hochgradig strukturierte Kommunikationslösung verzichtet werden. Zur Kommunikation der Entwickler untereinander wird der kollaborative, echtzeitfähige und webbasierte Texteditor EtherPad verwendet. Darin werden Textdateien für zu behebende Fehler, benötigte Funktionen, Diskussionen usw. gepflegt.

Programmierung

Im Folgenden wird die Implementierung der Aktivität *Dokumentieren* aus dem vorherigen Abschnitt in exemplarischen Ausschnitten beschrieben. Die jeweils dazugehörigen Quelltexte sind im Anhang zu finden.

Anfordern der Vorbelegungsdaten Aus den Studienmetadaten und dem Teilnehmerpseudonym wird eine SOAP-Nachricht generiert, die an den Webservice des KIS-Adapters gesendet wird. Die semantischen Annotationen aus dem Alias-Bereich von ODM werden genutzt, um dem KIS-Adapter mitzuteilen, welche klinischen Daten benötigt werden. Das Generieren der Nachricht erfolgt mittels eines XSLT-Programms. Zur Kommunikation mit dem Webservice wird die in Abschnitt 5.1.4 entwickelte Komponente verwendet. Das Modul zum Anfordern der Vorbelegungsdaten und eine beispielhaft generierte Nachricht an den Webservice sind in Abschnitt A.4.1 gezeigt.

Integration der Vorbelegungsdaten Die Daten der vom Webservice gelieferten Antwort werden in den Vorbelegungsbereich der ODM-Datenstruktur kopiert. Dazu wird für jedes erhaltene Vorbelegungsdatum anhand der semantischen Annotationen geprüft, zu welchem `ItemData`-Element der ODM-Datenstruktur es gehört. In diesem Element wird ein `prepopDataItem`-Element im Namensraum `prepop` mit den Vorbelegungsdaten als Vendor Extension eingefügt, falls es noch nicht aus einer vorherigen Vorbelegung existiert. In das Element `prepopDataItem` werden die Bezeichnung des Quellformulars, seine eindeutige Instanzkennung, das Dokumentationsdatum und der dokumentierte Wert übernommen. Die Funktion ist in Abschnitt A.4.2 wiedergegeben.

Automatisch Dokumentieren Zum automatischen Dokumentieren werden die Daten aus dem Element `prepopDataItem` in das `Value`-Attribut des Elements `ItemData` kopiert. Eventuell bestehende Werte werden damit überschrieben. Zusätzlich wird die für das Element ggf. bestehende Signatur gelöscht und eine neue Signatur eingefügt. Diese enthält den Dokumentationsort, den Benutzer, der die automatische Dokumentation veranlasst hat, und das aktuelle Datum. Abschnitt A.4.3 zeigt den Programmcode.

Formulardarstellung Die Erzeugung des XForms-basierten Dokumentationsformulars geschieht zur Laufzeit mit einem XSLT-Programm, das Studienmetadaten und Teilnehmerdaten als Parameter enthält. Das Datenmodell des Formulars besteht aus dem ODM-Teilbaum unter `SubjectData` des entsprechenden Studienteilnehmers. Die dazu passende Darstellung wird anhand der Metadaten erzeugt. Dazu gehört die Bindung an die richtigen Datentypen, die XForms veranlassen, die passenden Eingabeelemente darzustellen und Konsistenzprüfungen vorzunehmen, und die Positionierung der Eingabeelemente auf dem Bildschirm. Die Verknüpfung von Eingabeelementen, die durch die ODM-Metadaten

bestimmt werden, mit dem Datenmodell des Formulars geschieht, indem die Eingabeelemente als Referenz den Zugriffspfad auf das zugehörige Element des Datenmodells als XPath-Ausdruck erhalten. XSLT-Template-Ausdrücke folgen der in der Studiendefinition angelegten Kette aus ODM `*Def-` und `*Ref`-Elementen: StudyEvent → Form → ItemGroup → Item. Bei den Template-Aufrufen wird ein XPath-Ausdruck übergeben, an den die jeweilige Elementidentifikation angehängt wird. So entstehen XPath-Ausdrücke für alle in den Metadaten enthaltenen Items.

Abschnitt A.4.4 zeigt das Template zur Erzeugung der Seite mit dem Dokumentationsformular.

5.2.5 Einführung und Nutzung

Auswirkungen auf den Studiendokumentationsprozess

Die Webanwendung x4T ist bisher in drei Datenerhebungsszenarien eingesetzt worden:

- Für ein *Register* [32], in dem die Daten eines Patientenkollektivs erfasst werden und zur Beantwortung von später zu bestimmenden medizinischen Fragestellungen zur Verfügung stehen.
- Für eine *Biobank* [KS09], in der Patienten entnommene Gewebeproben gesammelt werden und gemeinsam mit als relevant erachteten Daten für zukünftig zu bestimmende Untersuchungen aufbewahrt werden.
- Für eine *forschungsinitiierte Studie* [Suv12], die durch den Prüfarzt initiiert wird und im Hinblick auf eine bestimmte Fragestellung geplant und durchgeführt wird.

Bei den ausgewählten Anwendungsfällen handelt es sich um Beobachtungsstudien, bei denen die Behandlung des Teilnehmers nicht von der Studiendurchführung beeinflusst wird und die nicht zur Entwicklung von Pharmaprodukten dienen. Für derartige Studien sind strenge regulatorische Vorgaben [Eur02] u. a. hinsichtlich der *Validierung* von Anwendungen [PIC07; ME13] zu beachten. Diese Anforderungen sind von einem universitären Software-Prototypen, wie im Rahmen des Single-Source-Projekts entwickelt, nicht zu erfüllen. Obwohl die x4T-Software mandantenfähig ist und somit konzeptionell sowie seitens der Implementierung mehrere Studien unterstützt, wurden separate Instanzen für die drei Szenarien verwendet, um eine gegenseitige Beeinflussung auszuschließen.

Zur Unterstützung unterschiedlicher Studien ist x4T konfigurierbar ausgelegt, so dass in allen Szenarien derselbe Programmcode zum Einsatz kommt. Durch das Hochladen der ODM-Datei einer der jeweiligen Studien und die Konfiguration einiger Parameter wird die Instanz an den Einsatzzweck angepasst.

Für den umfangreichsten Einsatz von x4T im Rahmen des dermatologischen Registers *Pruritusdatenbank*[4] wurden die Auswirkungen der Einführung des x4T-Prozesses und des Einsatzes der x4T-Software [BFB+14] untersucht. Die Ergebnisse sind im Folgenden zusammengefasst: Die Daten der Pruritusdatenbank wurden vor Einführung des Single-Source-Ansatzes separat von der Behandlungsdokumentation auf papierbasierten Formularen erhoben und anschließend mit dem Tabellenkalkulationsprogramm Microsoft Excel[5] in einem studienspezifischen Datenformat digitalisiert und ausgewertet. Zum Zeitpunkt der Umstellung auf x4T im März 2012 waren bereits 2075 Studienteilnehmer vorhanden, die nach dem Dual-Source-Ansatz dokumentiert worden waren. Für diese wurde eine Datenmigration zu x4T vorgenommen. Nachdem das Studienpersonal mit den neuen Prozessen und der neuen Software vertraut gemacht worden war, erfolgte die Umstellung schlagartig zu einem Stichtag im März 2012. Bis September 2013 stieg die Teilnehmerzahl auf 3785, es wurden also 1710 neue Teilnehmer nach dem Single-Source-Verfahren mit x4T dokumentiert. Die Studie ist als langfristiges Register ohne derzeit geplantes Ende der Erhebung ausgelegt, in dem Daten von Patienten mit Juckreizbeschwerden gesammelt werden. Da die zu erhebenden Merkmale im Zuge der Umstellung auf x4T überarbeitet wurden, stieg deren Anzahl von 139 in der Excel-basierten Version auf 213 in der aktuellen, auf x4T basierenden Implementierung. Von diesen können 135 aus dem KIS automatisiert dokumentiert werden, 78 Elemente werden manuell erfasst. Eine Messung von jeweils 25 Dokumentationsvorgängen mittels der ursprünglichen Vorgehensweise und nach Umstellung auf Single-Source ergab die in Tabelle 5.1 dargestellten Zeiten, die das Studienpersonal mit Dokumentationsaufgaben verbringt. Durch die Einführung der x4T Software konnte also eine erhebliche Reduzierung des Dokumentationsaufwands erreicht werden.

Auswirkungen auf die Weiterentwicklung und Wartung

Bei der Umstellung auf x4T wurde zunächst nur das Basissystem mit Nutzerverwaltung und Formularmanagement realisiert und in Betrieb genommen. Auf diese Weise konnte das Studienpersonal bereits vor der Realisierung der übrigen

[4] http://pruritusdatenbank.de/

[5] https://office.microsoft.com/de-de/excel/

Tabelle 5.1: Für einen Dokumentationsvorgang im Mittel benötigte Zeit, Anzahl der Stichproben: $n = 25$.

	Dual-Source	Single-Source	Ersparnis
Mittelwert	1116 s	334 s	782 s (70,1 %)
Standardabweichung	185 s	83 s	

Funktionen mit der im täglichen Routinebetrieb benötigten Datenerfassung beginnen. Die zusätzlich benötigten Funktionen wie Statistiken und Export wurden in späteren Inkrementen produziert und in Betrieb genommen. Somit wurde die Wartungsphase der ausgelieferten Software früh erreicht und lief zeitgleich mit der Entwicklung noch ausstehender Funktionen ab. Je nach Art der Wartungsanfrage kann bei agiler Entwicklung unterschiedlich damit verfahren werden:

Stabilisierungs- und Korrekturanforderungen bedeuten, dass eine implementierte Funktion nicht zufriedenstellend arbeitet, dies aber durch Testen nicht vor der Inbetriebnahme erkannt wurde. Bei solchen Anforderungen wird zur Implementierungsphase der betreffenden Funktionen zurückkehrt, nötige Korrekturen werden durchgeführt und entsprechende Modifikationen und Erweiterungen der Testfälle für die nicht erkannten Fehler vorgenommen.

Optimierungen und Funktionserweiterungen lassen sich im Sinne der agilen Entwicklung als neue Anforderung auffassen und stellen einen neuen Sachverhalt in der Analysephase dar, für den dann die weiteren Phasen durchlaufen werden. Je nach Umfang und Zuordnung zu bereits bestehenden Funktionen werden deren Entwurfsdokumente und Implementierungen angepasst oder neue erstellt.

Eine im Rahmen von x4T durchgeführte Funktionserweiterung betrifft die Anzeige der Elemente eines Formulars innerhalb des Layouts auf dem Bildschirm. Zuerst wurde die dargestellte Gruppierung nur aus der ODM-Struktur abgeleitet und die Items einer ItemGroup wurden direkt untereinander dargestellt. Im Einsatz zeigte sich, dass damit der Platz auf dem Monitor nicht gut ausgenutzt wird und bei der Anwendung vermehrt gescrollt werden musste. Daher sollten bestimmte, durch die Studiendefinition einzeln festzulegende Elemente auch als Zellen einer Tabelle darstellbar sein. Die Studienmetadaten mussten also eine Erweiterung um die benötigten Informationen erfahren. Dazu erfolgte eine Erweiterung des ODM-XML Schemas im Namensraum `x4t`. Die Definition der ItemGroup-Elemente wurden dazu um die optionalen Attribu-

te DisplayMatrixRows und DisplayMatrixColumns ergänzt, mit denen die Zahl der Zeilen und Spalten angegeben wird. Die Definition der Verweise auf Item-Elemente wurde um die Attribute DisplayInRow und DisplayInColumn erweitert, mit denen die jeweilige Zelle, in der das Element platziert werden soll, angegeben wird. Programm 5.1 zeigt die Schemaerweiterung durch Definition der in ODM dafür vorgesehenen Erweiterungsgruppen.

```
1  <xs:schema targetNamespace="http://x4t.uni-muenster.de" xmlns="http://x4t.
       uni-muenster.de">
2    ...
3    <xs:attributeGroup name="ItemGroupDefAttributeExtensionDefinition">
4      <xs:attribute name="DisplayMatrixRows" type="xs:integer"/>
5      <xs:attribute name="DisplayMatrixColumns" type="xs:integer"/>
6    </xs:attributeGroup>
7
8    <xs:attributeGroup name="ItemRefAttributeExtensionDefinition">
9      <xs:attribute name="DisplayInRow" type="xs:integer"/>
10     <xs:attribute name="DisplayInColumn" type="xs:integer"/>
11   </xs:attributeGroup>
12   ...
13 </xs:schema>
```

Programm 5.1: Erweiterung von ItemGroup und Item, um eine gezielte Platzierung der Elemente eines Formulars zu ermöglichen.

Auf Ebene der Geschäftslogik müssen keine Änderungen vorgenommen werden, da das Datenmodell transformationsfrei übergeben wird. In der Präsentationsschicht müssen die Attribute bei der Erzeugung der Darstellung des XForms-Formulars berücksichtigt werden. Das XSLT-Programm zur Generierung des Formulars erzeugt dann eine HTML-Tabelle und bettet die Formularelemente in die Zellen der Tabelle ein.

Wenn die erweiterte Platzierung in einer Studie genutzt werden soll, wird die ODM-Datei entsprechend erweitert. Die Kompatibilität mit vorhandenen ODM-Dateien bleibt bestehen, da die vorgenommenen Erweiterungen, wie alle Vendor Extensions in x4T, optional sind.

Hierbei zeigt sich die Wartungsfreundlichkeit der hier vorgestellten XML-basierten Architektur: Wenige Redundanzen im Programmcode erleichtern die erweiternde Wartung, da keine Mapper oder Datenzugriffs- und Transportobjekte, die aus fachlicher Sicht identische Daten zwischen den Schichten aus technischen Gründen konvertieren, angepasst werden müssen. Erweiterungen des XML Schemas sind meist transparent für höhere Schichten, die davon nicht betroffen sind, da der komplette Teilbaum eines XML-Dokuments verarbeitet wird.

Vorgenommene Erweiterungen werden ebenso wie neu implementierte Funktionen in das Codeverwaltungssystem eingepflegt und automatisiert durch den

Integrationsserver getestet, so dass sie im jeweils aktuellen Versionsstand zur Auslieferung und Installation für den Echtbetrieb bereitstehen.

6 Bewertung

In diesem Kapitel wird ein Vergleich hinsichtlich der Leistungsfähigkeit zwischen dem herkömmlichen Ansatz der Softwarearchitektur, bei dem unterschiedliche Paradigmen innerhalb der Schichten verwendet werden, und der hier vorgestellten XML-basierten Architektur gezogen. Der Vergleich erfolgt auf konzeptioneller Ebene, um von unterschiedlichen Optimierungsgraden der zur Verfügung stehenden Komponenten zu abstrahieren. Im Anschluss werden einige Erkenntnisse aus der Entwicklung und dem Echtbetrieb von x4T präsentiert und werden weitere Domänen gezeigt, in denen ein Einsatz der hier gezeigten XML-basierten Architektur erfolgversprechend ist.

6.1 Charakteristika der XML-basierten Architektur

6.1.1 Statische Betrachtung

Die Hauptmotivation für die Verwendung der beschriebenen XML-basierten Architektur im Rahmen des Single-Source-Projekts liegt in der Vermeidung der konzeptionellen Brüche. Die Anforderung, den Standard CDISC ODM zu unterstützen, bedeutet, dass eine Repräsentation der Anwendungsdaten in diesem XML-Format vorhanden oder erzeugbar sein muss. Bei Verwendung der traditionellen Schichtenarchitektur mit relationalem DBMS und objektorientierter Geschäftslogik, wie in Abschnitt 2.2.5 gezeigt, erfordert dies einen weiteren Transformationsschritt, indem entweder die relationalen Daten oder die Datenobjekte der Geschäftslogik in ein XML-Format serialisiert werden müssen. Dieselbe Technik muss auch angewendet werden, wenn eine Schnittstelle für die Kommunikation mit Webservices benötigt wird. Eine weitere Transformation wird für die Erzeugung der für den Benutzer vorgesehenen Präsentationsschicht benötigt: Entweder entsteht ein weiterer zu überbrückender Paradigmenbruch, wenn diese bspw. mittels auf Schlüssel-Wert-Paaren basierenden HTML-Formularen entworfen wird. Oder aber es ist eine technische Transformation innerhalb desselben Paradigmas nötig, bspw. bei Verwendung eines Frameworks, das aus Objekten der Geschäftslogik ein aus Sicht der Entwickler objektorientiertes Formular generiert. Bei Verwendung der hier entworfenen

XML-basierten Architektur wird durchgängig dasselbe Datenmodell verwendet. Teilbäume der persistierten XML-Dokumente können in allen Schichten verwendet werden.

Die Entwurfsphase, während der in der traditionellen Architektur neben den Schichten auch die zugehörigen Mappingtechniken entworfen werden, fällt bei Verwendung der XML-basierten Architektur durch den Verzicht auf Mapper entsprechend schlanker aus. Während die Verwendung relationaler Konzepte in der Persistenzschicht und objektorientierter Konzepte in der Geschäftslogikschicht den Einsatz unterschiedlicher Modellierungstechniken, meist ERM-Diagramme und UML-Klassendiagramme, erfordert, so entfällt dieser Impedance Mismatch bei Verwendung der XML-basierten Architektur. Durch die Verwendung der Modellierungstechnik XML-Netze lassen sich Datenstrukturen, Funktionen und Prozesse integriert entwickeln. Die dabei zur Spezifikation der Objektspeicher dienenden XML Schemata bilden die Grundlage für die Implementierung der Persistenzschicht, die Filterschemata zur Spezifikation der Objekttransformationen stellen die Grundlage für die Implementierung der Geschäftslogik dar.

In der Implementierungsphase wird mit einem über die Schichten der Software hinweg einheitlichen Datenmodell gearbeitet. Es müssen keine Hilfstechniken wie Mapper oder Datenzugriffsobjekte erstellt oder konfiguriert werden. Insbesondere für hierarchische Datenstrukturen, die durch eine Hierarchie von Klassen mit 1:n-Assoziationen abgebildet werden, müssen eine große Zahl Klassen, deren Hauptzweck die Bereitstellung der Anwendungsdaten ist, und Hilfstechniken entworfen und implementiert werden. Ein Beispiel dafür ist die bereits in Abbildung 2.7 auf Seite 36 gezeigte Struktur der klinischen Daten in ODM. Bei Verwendung der XML-basierten Architektur sind derartige Konstrukte überflüssig. Ferner erfordert das Implementieren von fachlichen Funktionen über mehrere Schichten kein Umdenken bzgl. des Paradigmas, sondern erlaubt die Verwendung des gleichen Modells. Ein agiles Vorgehen mit gemeinsamem Codebesitz und das Arbeiten an einzelnen fachlichen Funktionen wird dadurch erleichtert, dass weniger Spezialkenntnisse einer Schicht für die Arbeit an Funktionen erforderlich sind.

Für den sich während des Betriebs der Anwendung manifestierenden Wartungsbedarf sind die Vereinfachungen des Entwurfs und der Implementierung ebenfalls relevant. Das gemeinsame Datenmodell erleichtert Änderungen, da keine ggf. zu modifizierenden Mapper existieren. Entwurfsdokumente und Code können somit einfacher angepasst werden. Es existiert allerdings eine starke Abhängigkeit von den zugrunde liegenden XML Schemata. Da diese vorzugsweise

auf veröffentlichten und stabilen Standards einer Domäne, die einen fachlichen Hintergrund haben, basieren, ist diese Abhängigkeit aber ohnehin aus fachlicher Sicht gegeben. Bei traditioneller Architektur ist die direkte technische Abhängigkeit höherer Schichten vom Datenmodell der Persistenzschicht geringer, allerdings müssen bei fachlichen Änderungen daran auch Änderungen der übrigen Schichten erfolgen, damit die Modifikationen in der Webanwendung genutzt werden können.

Neben diesem qualitativen Unterschied, der sich hauptsächlich auf Entwurf und Implementierung der Software bezieht, soll eine Betrachtung der sich zur Laufzeit ereignenden Vorgänge stattfinden.

6.1.2 Betrachtung der Laufzeit

Der empirische Geschwindigkeitsvergleich von Applikationen, die nach dem traditionellen Entwurf auf relationalen und objektorientierten Komponenten basieren, und von solchen, die mittels der hier vorgestellten XML-basierten Architektur implementiert sind, ist problematisch: So ist es bei einer empirischen Vergleichsmessung aufgrund der unterschiedlichen Konzepte und Techniken nicht möglich, konzeptionelle von implementierungsbedingten Einflüssen zu unterscheiden. Am Programmcode beider Applikationen sind beliebige, schwer zu identifizierende Optimierungen möglich und die Reife der eingesetzten Softwarekomponenten ist kaum zu bestimmen, jedoch aufgrund der historischen Entwicklung recht unterschiedlich: Während relationale DBMS schon seit Anfang der 1980er Jahre kommerziell verfügbar und stark verbreitet sind [Vos08], sind XML-DBMS erst um die Jahrtausendwende aufgekommen [GG01] und haben in praxi noch keine so starke Verbreitung erfahren. Daher wird im Folgenden eine konzeptionelle Bewertung vorgenommen, die von konkreten Implementierungen und somit Optimierungsmöglichkeiten abstrahiert. Zur Abschätzung der oberen Schranken der Laufzeit wird die O-Notation [Knu76] verwendet.

Betrachtet wird die Komplexität des Lesens, da dies die häufigste Operation ist, die u. a. für das Anzeigen von Formularen, die Erzeugung von Statistiken und den Datenexport benötigt wird. Bei Verwendung der traditionellen Architektur mit relationalem DBMS in der Persistenzschicht und Objektorientierung in der Geschäftslogik stellt Vererbung eine Herausforderung dar, für die die in Abschnitt 2.3.1 erläuterten Mapping-Strategien *table-per-class*, *table-per-concrete-class* und *table-per-class-family* zur Verfügung stehen.

Der table-per-class-Ansatz bildet als einziger der drei Ansätze die Objektstruktur redundanzfrei und ohne zwangsläufig nullwertige Attribute ab und bie-

tet diese aus Sicht des Datenbankentwurfs wünschenswerten Eigenschaften. Er erfordert aber für jede Subklasse eine Join-Operation beim Lesen der Objekte.

Zur Veranschaulichung einer einfachen Vererbung seien R und S Relationen, die eine vererbende und eine erbende Klasse repräsentieren und die n und m Tupel aufweisen. Joins über mehrere Relationen werden durch die Hintereinanderausführung von Joins über je zwei Relationen gebildet [ML86]. Daher tritt die Problemstellung bei umfangreicheren Vererbungshierarchien je zusätzlicher Relation erneut auf.

Für Join-Operationen stehen unterschiedliche Algorithmen zur Verfügung, die jeweils eine unterschiedliche Komplexität aufweisen. Wichtige Algorithmen sind *Nested-Loop-Join*, *Sort-Merge-Join* und *Hash-Join* [Vos08].

Nested-Loop-Joins stellen die einfachste Art der Ausführung dar. Dabei wird in zwei geschachtelten Schleifen über die Tupel von R und S iteriert und für jede Tupelkombination verglichen, ob sie identische Join-Attribute aufweisen. Ist dies der Fall, wird die Kombination in die Ergebnisrelation übernommen. Nested-Loop-Join weist eine Komplexität von $O(m \cdot n)$ auf [ME92].

Sort-Merge-Join läuft in zwei Phasen ab und erzielt einen Geschwindigkeitsgewinn durch Reduzieren der Vergleichsoperationen zwischen den Tupeln. In der ersten Phase werden die Tabellen bzgl. der Join-Attribute sortiert. In einer zweite Phase werden die sortierten Tabellen durchlaufen und bei identischem Join-Attribut wird ein Tupel der Ergebnisrelation gebildet. Die Komplexität wird dabei durch die Sortierfunktion bestimmt, die üblicherweise $O(n \log n)$ je Relation beträgt [ME92].

Hash-Join reduziert die Laufzeitkomplexität weiter, indem statt der Sortierung ein Hashingverfahren angewendet wird. Zunächst wird auf das Join-Attribut jedes Tupels der Relation S eine Hashfunktion angewendet und das Tupel wird in einer Hashtabelle gespeichert. Dann wird für die Join-Attribute jedes Tupels der Relation R mit derselben Hashfunktion geprüft, ob in der Hashtabelle von S ein Eintrag vorhanden ist. Wenn dies der Fall ist und die Join-Attribute identisch sind, wird ein Tupel für die Ergebnisrelation gebildet. Die Komplexität des Hash-Join ist $O(n + m)$, also linear, da beide Relationen einmal durchlaufen werden [ME92].

Im besten Fall können Objekte, die in eine Vererbungsstruktur eingebunden sind, also in linearer Zeit aus den relationalen Daten rekonstruiert werden. Dieser Schritt entfällt bei Verwendung von XML-Daten in Persistenz- und Geschäftslogikschicht.

XML-Dokumente weisen eine Baumstruktur auf, die von XML-DBMS als logisches Datenmodell verwendet wird. XML-DBMS nutzen je nach Imple-

mentierung unterschiedliche Vorgehensweisen zum Serialisieren von XML-Dokumenten. eXist-db nutzt zur Nummerierung der Knoten eines Dokumentes einen modifizierten k-nären Baum und bildet zur Suche von Elementen einen B^+-Baum [Mei03]. Die Suche in einem B^+-Baum erfordert logarithmischen Aufwand, also $O(\log n)$ [Com79].

Konzeptionell kann ein XML-Dokument bzw. ein Teilbaum davon also schneller als eine nach dem table-per-class-Ansatz persistierte Objektstruktur gelesen werden.

6.2 x4T

Die Motivation für die Entwicklung von x4T besteht in der Erprobung des Single-Source Ansatzes in der klinischen Forschung. Die aus fachlicher Sicht bestehenden Defizite des Dual-Source-Verfahrens, nämlich die redundante Datenerhebung und die dadurch bedingte ineffiziente Ausführung des Dokumentationsprozesses, sollen behoben werden. Aus Sicht der Studiendurchführenden besteht ein hohes Interesse an der Optimierung der Prozesse durch Einführung des Single-Source-Konzepts. Wie in Abschnitt 5.2.5 dargestellt, konnte die Durchführung der Dokumentation um ca. 70 % beschleunigt werden, womit der Nutzen für die Anwender deutlich ist.

Die dazu verwendete Technik der XML-basierten Architektur steht nicht direkt im Blickfeld der Anwender. Die von den Anwendern genannten grundlegenden Anforderungen implizieren aber einige dafür günstige Voraussetzungen, wie im Folgenden argumentiert wird.

So soll die Anwendung aus Sicht der Benutzer in den klinischen Arbeitsablauf integriert werden. Dies bedeutet, dass die Anwendung an den üblichen Arbeitsplätzen zur Verfügung stehen muss und legt die Realisierung als Client-Server-Modell nahe. Die Integration in den Arbeitsablauf bedeutet ebenfalls, dass sie aus dem zur Behandlungsdokumentation genutzten KIS zugreifbar sein muss. Für die Entwicklung bietet es sich daher an, KIS-spezifische Teile der Anwendung von den unspezifischen Teilen zu trennen und die Komponenten durch standardisierte Schnittstellen zu verbinden. Für x4T bedeutet dies die Aufteilung in KIS-Adapter, der in das KIS integriert wird, und Formularmanagement, das als Webanwendung zur Verfügung steht. Die Kommunikation über Webservices liegt nahe, da diese Interoperabilität über Plattformgrenzen hinweg vorsehen und somit die Implementierung eines KIS-Adapters unabhängig von der des Formularmanagements möglich ist. Die Forderung nach Vermeidung eines Lock-in-Effekts bezüglich der Daten bedeutet, dass diese in ein standardisier-

tes Format exportierbar sein müssen. Hierzu bietet sich für klinische Studien das XML-basierte CDISC ODM-Format an, da es anders als ein domänenunabhängiges Format wie CSV auch den vollständigen Export der Metadaten erlaubt. Der grundlegende Zweck der Anwendung, die Erfassung und Bearbeitung stark strukturierter Daten, erfordert die Repräsentation als Formular. Gerade für umfangreiche Datensätze bietet sich hierzu XForms an.

Das Konzept der hier vorgestellten XML-basierten Architektur erleichtert die Erfüllung vieler an x4T gestellter Anforderungen, die bei Implementierung mit einer anderen Architektur ggf. aufwändige Implementierungen nach sich ziehen.

Das agile Vorgehen bei der Entwicklung erlaubt ein zielgruppengerechtes Spezifizieren der Software. Vor Einsatz der Software fällt es den befragten Anwendern meist schwer, die gewünschten Funktionen vollständig zu spezifizieren. Es fällt ihnen hingegen leichter, anhand der im Einsatz befindlichen Software festzustellen, welche zusätzliche Funktionen noch benötigt werden oder welche Änderungen ggf. vorzunehmen sind. Verbunden mit dem Interesse an schnellem Einsatz der Anwendung legt das iterative Spezifizieren von Funktionen ein agiles Vorgehen nahe. Dieses passt ebenfalls gut zu der hier gezeigten XML-basierten Architektur. Der Verzicht auf Transformationskomponenten stärkt die Konzentration auf die aus fachlicher Sicht Nutzen stiftenden Komponenten, deren Entwicklung agil durch den Anwender bestimmt wird. Die agile Entwicklung erlaubt die schnelle Umsetzung von Änderungen, Wartungs- und Pflegeanforderungen sowie deren Inbetriebnahme durch häufiges Deployment einer neuen Softwareversion. Das XML-Format ist gut automatisiert zu verarbeiten, was die Erstellung und Ausführung von Testfällen begünstigt. Deren automatische Ausführung ist bei agiler Entwicklung besonders wichtig.

Die agile Entwicklung von x4T als XML-basierte Architektur ist also technisch und organisatorisch der Domäne der klinischen Forschung angemessen.

6.3 Weitere Anwendungsbereiche

Die hier vorgestellte XML-basierte Architektur eignet sich besonders für Webanwendungen, die mit strukturierten, als Dokumente organisierten Daten arbeiten. Vorteilhaft ist es, wenn in der Domäne XML-basierte Standards verbreitet sind. Die folgenden Abschnitte skizzieren solche Anwendungsfälle.

6.3.1 Open Data

Bei den unter dem Schlagwort *Open Data* zusammengefassten Aktivitäten handelt es sich um das Bereitstellen von Daten für die Allgemeinheit. Die Open Knowledge Foundation (OKF) definiert Open Data zusammengefasst wie folgt:

> „A piece of data or content is open if **anyone** is **free to use**, **reuse**, **and redistribute** it — subject only, at most, to the requirement to attribute and/or share-alike." [SDH+09]

Die Bereitstellung der Daten muss also für jedermann und zur freien Verwendung, Verarbeitung und Weitergabe erfolgen.

Insbesondere die öffentliche Verwaltung verfügt über vielerlei Daten, die im Rahmen von Informationsfreiheit und Open Government als Open Data verfügbar gemacht werden können oder auf Grund gesetzlicher Bestimmungen verfügbar gemacht werden müssen. Diese werden von den Behörden unter eigenen Webseiten bereitgestellt, bspw. für die Bundesrepublik Deutschland unter *GovData*[1] und für die USA unter *Data.gov*[2]. Es existiert eine steigende Zahl weiterer Portale anderer Länder. Das Portal *datacatalogs.org*[3] stellt eine Suchfunktion für eine Reihe von Open Data-Webseiten zur Verfügung. Die Bereitstellung von Open Data ist nicht auf Behörden beschränkt. Auch wissenschaftliche Organisationen, private Initiativen und Unternehmen können ihre Daten so zur Verfügung stellen. Die Europäische Kommission sieht in Open Data eine Möglichkeit zur Wertschöpfung und betont den Bedarf an technischen Mitteln zum Umgang mit Open Data [Sof12]. Der Wissenschaftsbetrieb kann ebenfalls durch den Zugriff auf Open Data profitieren, da hierdurch Kosten der Forschung gesenkt und neue Erkenntnisse schneller gewonnen werden können [US07; Mur08]. Da das öffentliche Interesse an Open Data und der dadurch ermöglichten Wertschöpfung wächst [Aut13], ist davon auszugehen, dass langfristig erhebliche Mengen und relevanter Daten zur Verfügung stehen.

Wichtig für die Nutzung dieser Daten ist die strukturierte Bereitstellung, um Maschinenlesbarkeit und somit Weiterverarbeitung zu ermöglichen [BETL12]. Das XML-Format ist als aufgrund seiner Eigenschaft, Daten und Metadaten gemeinsam zu enthalten, ein geeignetes Transportmedium für Open Data. Folgerichtig werden viele Open Data-Angebote in XML-basierten Formaten zur Verfügung gestellt. XML ist mit derzeit ca. 12 000 Einträgen das am häufigsten verwendete Format des Portals Data.gov.

[1] https://www.govdata.de/

[2] http://www.data.gov/

[3] http://datacatalogs.org/

Insofern erscheint die Nutzung der hier vorgestellten XML-basierten Architektur für Softwaresysteme, die Open Data nutzen oder bereitstellen, naheliegend. Über Open Data-Portale verfügbare XML-Daten können damit ohne Transformation gelesen, weiterverarbeitet und gespeichert werden. Seitens der Datenanbieter steht damit ein Konzept zur Verfügung, das das XML-Format ggf. schon ab der Erfassung im Formular unterstützt und eine weitere Verarbeitung und Publikation ohne Umwandlung in andere Formate erlaubt. Zu untersuchen bleibt, ob die zur Verfügung stehenden Implementierungen der XML-Komponenten, insbesondere der DBMS, den oftmals großen Datenmengen von Open Data gewachsen sind.

6.3.2 Klinische Domäne

Neben dem gezeigten Anwendungsbeispiel x4T, das der klinischen Forschung dient, bietet die klinische Domäne weitere Einsatzgebiete für hier vorgestellte XML-basierte Architektur. Das üblicherweise für die Kommunikation von Softwareprogrammen unterschiedlicher Hersteller verwendete Protokoll im Gesundheitswesen ist der *Health Level 7 (HL7)*[4] *Messaging Standard*. Die bislang im Praxiseinsatz befindliche Version 2.x ist stringbasiert und verfügt über kein konsistentes, semantisches Referenzmodell. Der Nachfolger Version 3 [Hin07] basiert auf einem semantischen Referenzmodell und nutzt XML-Nachrichten.

Basierend auf HL7 Version 3 existiert mit HL7 *Clinical Document Architecture (CDA)* [DAB+05] ein Format zur strukturierten Darstellung klinischer Dokumente [Ben12]. Damit können bspw. Arztbriefe, Befundungen, Impfpässe, Formulare zur Übermittlung meldepflichtiger Krankheiten usw. strukturiert repräsentiert werden.

Die semantische Unterstützung ist in drei Graden, genannt Level, vorgesehen. Level 1 umfasst den geringsten Grad der Strukturierung und dient der korrekten Wiedergabe menschlich lesbarer klinischer Dokumente. Dazu wird eine HTML-ähnliche Strukturierungssprache verwendet, mittels derer Dokumente bestehend aus Abschnitten, Texten, Tabellen, Bildern etc. repräsentiert werden.

Level 2 erweitert diese Repräsentation um abschnittsweise maschinenlesbare Beschreibung der Inhalte. Es kann also bspw. hinterlegt werden, dass es sich bei einem Dokument um einen Arztbrief handelt, der Befunde einer körperlichen Untersuchung, Laborwerte und Medikation enthält. Ein System, das ein Dokument des Levels 2 interpretieren kann, ist also in der Lage, die Bedeutung einzelner Bestandteile des Dokuments zu erkennen und weiterzuverarbeiten.

[4] http://www.hl7.org/

Dies erleichtert die Informationsverarbeitung, da die Suche nach bestimmten Kriterien, wie bspw. nach allen Unterlagen mit Laborwerten eines Patienten, automatisiert durchführbar ist.

Bei Dokumenten des Levels 3 werden zusätzlich einzelne Elemente semantisch ausgezeichnet. Diagnosen werden mittels eines Vokabulars codiert und jeder einzelne Laborwert wird maschinenlesbar annotiert. Dies stellt die höchste Stufe der Strukturierung dar und erlaubt umfangreiche Anwendungen. So ist es möglich, bei der Überschreitung von Normwerten automatisch Hinweise o. ä. auszulösen, bei Verordnung von Medikamenten auf dokumentierte Allergien gegen Inhaltsstoffe oder sonstige Kontraindikationen zu prüfen oder Daten der Behandlungsdokumentation ohne weitere Codierungen für wissenschaftliche Zwecke oder andere Sekundärverwendungen zur Verfügung zu stellen.

CDA-Dokumente sind ebenfalls XML-basiert und müssen erstellt, gespeichert, übertragen und verarbeitet werden. Für diesen Zweck scheint die hier vorgestellte XML-Architektur ideal geeignet. Ein XForms-basierter Dokumenteneditor erlaubt die Erzeugung eines XML-Dokuments, ein XML-DBMS erlaubt Speichern, Lesen, Ändern und Suchen der standardisierten Dokumente, mittels XQuery lassen sich beliebige Analysen auf den Originaldokumenten ausführen und für die Kommunikation mit anderen Systemen liegt das Dokument bereits im dafür vorgesehenen auf Interoperabilität ausgelegten Format vor.

6.3.3 Medienverwaltung

Bibliotheken verwalten ihren Medienbestand anhand von strukturierten und detailliert erfassten Metadaten. Interoperable Formate erlauben es, Metadaten nicht nur im ursprünglichen Kontext, sondern auch über System- und Organisationsgrenzen hinweg zu nutzen, was eine wünschenswerte Eigenschaft darstellt [Keß08]. Die Deutsche Nationalbibliothek[5] verwendet bspw. Metadaten in Formaten MARC21 und DNB Casual [19]. *MARC21* [FF03; 22] dient als internationales, einheitliches Austauschformat für die Kommunikation mit anderen Bibliotheken. Es enthält umfangreiche Möglichkeiten, Metadaten von Medien strukturiert anzugeben. MARC21-Datensätze können auch zur weiteren Nutzung von der Nationalbibliothek käuflich erworben und dann in der XML-Repräsentation heruntergeladen werden. Weniger umfangreiche und kostenfrei erhältliche Metadaten zu Medien werden im Format *DNB Casual* angeboten. Dieses Format basiert auf einer Teilmenge des als *Dublin Core*[6] standardisierten

[5] http://www.dnb.de/

[6] http://dublincore.org/schemas/xmls/

Metadatenformats und wird ebenfalls als XML-Dokument angeboten.

Der Bereich der Verwaltung von Medien, wie in einer Bibliothek, oder anderer Objekte mit strukturierten Metadaten stellt ein weiteres prädestiniertes Einsatzgebiet für die hier präsentierte XML-basierte Architektur dar. Die Erfassung von Daten kann automatisiert per Webservice oder manuell in einem XForms-Formular direkt im XML-Format erfolgen. Bei Verwendung der standardisierten Formate ist die Semantik der Datenelemente einheitlich gekennzeichnet, so dass Datenimport und Export einfach realisierbar sind. Außerdem lassen sich Funktionen der Geschäftslogik als XQuery-Module entwickeln, die in unterschiedlichen Systemen Anwendung finden können, ohne auf ein proprietäres, internes Datenformat Rücksicht nehmen zu müssen. XSLT-Programme können genutzt werden, um Umwandlungen in andere, insbesondere XML-basierte, Formate zu erzeugen, wie das Exportformat DNB Casual im Beispiel oder die XHTML-basierten Seiten eines Online-Katalogs.

6.3.4 Finanzbranche

In der Finanzbranche ist der Austausch von Informationen mittels strukturierter Nachrichten weit verbreitet. Für den weltweiten Austausch von Zahlungsinformationen stellt die *Society for Worldwide Interbank Financial Telecommunication (SWIFT)* ein Kommunikationsnetzwerk bereit und definiert Nachrichtenarten. Die dabei verwendeten Nachrichtenformate werden als *Message Types (MTs)*[SWI13a] bezeichnet und sind historisch bedingt textbasiert und nutzen Schlüssel/Wert-Paare zur Strukturierung der Nachrichten. Zu diesen Formaten kommen in jüngerer Zeit verstärkt XML-basierte Nachrichtentypen, *Standards MX messages (MX)* [SWI13b] genannt, hinzu, bzw. lösen sie ab. Die neueren, XML-basierten Nachrichten sind als ISO 20022 [ISO13] genormt.

Bei der Verwendung ist ein Trend zugunsten der XML-basierten Nachrichten des ISO 20022 Standards zu beobachten. Bspw. basieren Zahlungen im *Single Euro Payments Area (SEPA)*-System darauf [San12], für das europäische Zahlungssystem der Zentralbanken *TARGET2* ist die Umstellung im November 2017 vorgesehen [Din13] und die Umstellung von weltweitem *High-Value-Payment* ist ebenfalls geplant [BL12].

Neben den als ISO 20022 spezifizierten Nachrichtenarten kommen noch weitere XML-basierte Nachrichtenarten in der Finanzbranche zum Einsatz, darunter bspw. das Nachrichtenformat *Financial Information eXchange Markup Language (FIXML)* [FPL13] für den Handel mit Wertpapieren oder *Financial products Markup Language (FpML)* [FpM14] als Beschreibungssprache für Finanzprodukte.

Die Verwendung der hier vorgestellten XML-basierten Architektur zur Implementierung von Software in dieser Branche erscheint attraktiv, weil mit Einführung der moderneren Nachrichtenarten XML-basierter Datenaustausch vorliegt und diese Nachrichten dann mit nativen Sprachmitteln erzeugt, validiert, gespeichert und verarbeitet werden können. Insbesondere bei großen Datentransaktionsvolumina ist der Verzicht auf Transformationen ein konzeptioneller Vorteil. Für eine Implementierung ist zu prüfen, ob die relativ jungen Implementierungen der XML-Komponenten einen ausreichenden Reifegrad aufweisen, um die etablierten Softwarekomponenten zu ersetzen.

6.3.5 Immobilienbranche

In der Immobilienbranche sind Objekte hauptsächlich durch strukturierte Merkmale charakterisiert. So dient eine Immobilie einem bestimmten Zweck, wird zu einem bestimmten Preis zur Miete oder zum Kauf angeboten und verfügt über Eigenschaften wie Fläche, Baujahr, Anzahl an Räumen, Adresse usw. Im deutschsprachigen Raum wird zur strukturierten Beschreibung dabei *OpenImmo*[7] als Standard verwendet, im englischen Sprachraum kommen die durch das *Open Standards Consortium for Real Estate (OSCRE)* definierten Standards zum Einsatz[8] [Spä14]. Diese Standards basieren auf XML und stellen entsprechende Schemata bereit. Im Standard OpenImmo sind etwa 300 Merkmale definiert, die Standards der OSCRE enthalten insgesamt noch mehr. Der primäre Anwendungszweck liegt auf dem Datenaustausch zwischen den Handelsteilnehmern und Plattformbetreibern des Immobilienmarktes. Viele Plattformen sind als Webanwendungen realisiert und werden sowohl von professionellen Maklern als auch privaten Käufern und Verkäufern genutzt. Für professionelle Anwender bieten die Plattformen die Möglichkeit zum Datenimport in einem der standardisierten oder einem anbieterspezifischen Format. Somit muss der Makler diese Daten nur einmal erfassen und kann sie automatisch zu den gewünschten Plattformen exportieren. Privatanwender nutzen die Anwendung über die Weboberfläche zum Suchen nach Inseraten anhand bestimmter Kriterien.

Auch für Software in dieser Domäne scheint die Verwendung der hier vorgestellten XML-basierten Architektur angebracht. Für die Datenerfassung durch Makler bieten sich Formulare auf XForms-Basis an, mittels derer die Immobiliendaten ohne Brüche im Zielformat erfasst und über Webservices an die Schnittstellen der Plattformbetreiber geschickt werden können. Dazu ggf. nötige Trans-

[7] http://www.openimmo.de/
[8] http://www.oscre.org/

formationen in andere XML Schemata lassen sich durch XSLT realisieren. Für Plattformbetreiber bietet sich diese XML-basierte Architektur ebenfalls an. Sie müssen große Mengen importierter Daten verarbeiten und speichern. Die importierten Daten können transformationsfrei in einem XML-DBMS gespeichert werden und können per XQuery durchsucht werden. Neben der Suche basierend auf bestimmten Kriterien dürfte das Anzeigen von einzelnen Immobilieninseraten der Hauptanwendungsfall einer solchen Plattform sein. Dies passt gut zu der Dokumentenorientierung von XML, die einen Zugriff auf einzelne Dokumente erleichtert. Die Dokumente können durch Anwenden von Query- und XSLT-Programmen zur HTML-Darstellung im Browser gerendert werden.

7 Schlussbetrachtung

Im Rahmen der traditionellen Schichtenarchitektur für Webanwendungen treten Brüche im Datenmodell zwischen relationaler Datenhaltung, objektorientierter Geschäftslogik, XML-basierten Webservices und HTML-basierter Präsentationsschicht auf. Insbesondere der Impedance Mismatch zwischen relationalem Datenmodell und Objektorientierung ist problematisch, da alle zur Verfügung stehenden Transformationen spezifische Nachteile aufweisen, die sich aus den unterschiedlichen Paradigmen ergeben.

Da die innerhalb einer Schicht verwendeten Konzepte und Implementierungen sehr ausgereift sind, werden die Brüche beim Softwareentwurf in Kauf genommen und durch den Einsatz von Transformationstechniken überbrückt, aber nicht konzeptuell beseitigt.

Die in der vorliegenden Arbeit präsentierte XML-basierte Architektur greift auf die in jüngerer Zeit entstandenen Spezifikationen der XML-Technikfamilie und deren Implementierungen zurück. Aus diesen auf dem XML-Standard aufbauenden Bestandteilen wird eine Softwarearchitektur gebildet, die den Einsatz von Transformationstechniken vermeidet und ein in allen Schichten einheitliches Datenmodell verwendet. Die prinzipielle Tragfähigkeit dieser Architektur konnte durch den Einsatz im Rahmen des Single-Source-Projekts gezeigt werden.

Als Entscheidungshilfe, ob zur Realisierung eines Projekts zur Erstellung einer Webanwendung von der XML-basierten Architektur Gebrauch gemacht werden sollte, werden die Charakteristika in Tabelle 7.1 als SWOT[1]-Matrix illustriert. Diese teilt die entscheidenden Einflüsse in interne und externe sowie positive und negative auf. Interne Einflüsse liegen im Gegenstand der Betrachtung begründet, externe Einflüsse entstehen durch das Umfeld. Es ergeben sich somit vier Felder: Interne positive Einflüsse sind *Stärken*, interne negative Einflüsse heißen *Schwächen*, externe positive Einflüsse werden als *Chancen* bezeichnet und bei externen negativen Einflüssen handelt es sich um *Risiken* [KBB10].

[1]Strengths, Weaknesses, Opportunities, Threats

Tabelle 7.1: SWOT-Matrix mit Einflussfaktoren auf den Einsatz der XML-basierten Architektur.

Stärken	Schwächen
▪ auf fachliche Aspekte konzentrierte Architektur ▪ passt gut zu agiler Entwicklung ▪ basiert auf abgestimmten Standards ▪ einheitliches Datenmodell	▪ Reife der Komponenten ▪ keine langfristige Erfahrung ▪ bei schwach strukturierten Daten
Chancen	**Risiken**
▪ Weiterentwicklung von domänenspezifischen XML Schemata ▪ weitere Datenquellen über öffentliche Webservices und im Rahmen von Open Data zu erwarten	▪ Zukunft einzelner Komponenten ▪ Verfügbarkeit von Experten ▪ Ablösung von XML als Lingua Franca des Internets

Stärken Neben dem Einsatz in der klinischen Forschung eignet sich die XML-basierte Architektur besonders dann, wenn Anwendungen Daten verarbeiten, die in Form von Dokumenten strukturierbar sind. Häufig liegen für solche Dokumente bereits anerkannte XML Schemata vor. Mit XML-Netzen liegt ein Modellierungswerkzeug vor, das sich aufgrund seines Datenmodells nahtlos gut für den Softwareentwurf im Rahmen der XML-basierten Architektur eignet. Die Softwareentwicklung durchläuft die bekannten Phasen Analyse, Entwurf, Implementierung sowie Einführung und Nutzung, die entweder mittels einer herkömmlichen Methode hauptsächlich seriell oder mit einer agile Methoden stark iterativ stattfinden können. Die Architektur nutzt XML in allen Schichten und vermeidet die sonst üblichen Brüche und damit verbundene Transformationskomponenten. In allen Schichten kommen dabei native, vom W3C standardisierte und aufeinander abgestimmte XML-Techniken zur Speicherung, Verarbeitung und Darstellung zum Einsatz. Die Entwicklung ist stärker auf die fachlichen, d. h. dem Anwender Nutzen stiftenden, Aspekte konzentriert, da kei-

ne Transformationskomponenten entworfen und gepflegt werden müssen. Dies passt gut zu agilen Entwicklungsverfahren, da bei diesen die Umsetzung von Funktionen vom Nutzen für den Abnehmer der Software bestimmt wird.

Schwächen Für den praktischen Einsatz spielt neben den konzeptionellen Eigenschaften der Architektur die Verfügbarkeit von ausgereiften und bewährten Softwarekomponenten eine wichtige Rolle. Hier weisen die traditionellerweise verwendeten relationalen DBMS, objektorientierten Programmiersprachen und HTML-Frameworks eine lange Historie und wohl einen höheren Optimierungsgrad als die relativ jungen Implementierungen der XML-Familie auf. Über den Betrieb liegen in Literatur und bei den meisten Anwendern umfangreichere Erfahrungen vor, die für die XML-basierten Komponenten noch gesammelt werden müssen. Die Akzeptanz des innovativen Ansatzes kann daher bei konservativ eingestellten Anwendern eingeschränkt sein.

Eine weitere Schwäche liegt in der Verarbeitung schwach strukturierter Daten. Für Anwendungen, deren Zweck bspw. in der Verarbeitung auf Bild- oder Tondaten liegt, ist die XML-basierte Architektur weniger geeignet.

Chancen Geänderte Anforderungen innerhalb der Anwendungsdomäne führen zur Weiterentwicklung der domänenspezifischen XML Schemata. Die XML-basierte Architektur bietet die Chance, an diesen Änderungen zu partizipieren, ohne ein eigenes Datenmodell entwickeln zu müssen. Abwärtskompatibel weiterentwickelte XML Schemata erlauben die sukzessive Einführung von Neuerungen, die nach Nutzen für den Anwender priorisiert werden können.

Weitere Maßnahmen im Rahmen von Open Data, die durch Gesetze vorgeschrieben oder auf Grund von unternehmerischer Überlegung durchgeführt werden, lassen zukünftig noch mehr Datenquellen für strukturierte Daten erwarten. Die Anwendung der XML-basierten Architektur ist bei dieser Art von Daten besonders günstig.

Risiken Ein Risiko stellt die mögliche Einstellung der Weiterentwicklung ausgewählter Komponenten dar. Da die Verbreitung geringer als bei traditionellen Komponenten ist und die dahinterstehenden Unternehmen relativ klein sind, ist dieses Risiko größer als bei traditioneller Architektur. Durch die Standardisierung der Komponenten werden die Auswirkungen in diesem Fall allerdings gemindert, da die Austauschbarkeit erleichtert ist. Das Risiko kann durch die Verwendung von Komponenten unter Open Source-Lizenz weiter gemindert werden, da dann eine Weiterentwicklung auch ohne den ursprünglichen Ersteller

möglich ist.

Mit der derzeit geringeren Verbreitung von XML-Komponenten geht einher, dass nicht so viele Experten für das Gebiet der XML-Technikfamilie wie für populärere Techniken verfügbar sind und es somit zu Engpässen an geeigneten Entwicklern kommen könnte.

Ein weiteres Risiko bei Einsatz der XML-basierten Architektur ist die Verdrängung von XML als Lingua Franca des Internets durch eine Konkurrenzentwicklung. In diesem Fall würden weder domänenspezifische XML Schemata noch die W3C Standards und dazugehörige Implementierungen weitere Entwicklung erfahren.

Handlungsempfehlung Auf Grund der vielen zu berücksichtigenden Aspekte bei der Frage nach der richtigen Softwarearchitektur für eine Webanwendung kann keine allgemeingültige Entscheidungsregel aufgestellt werden. Abschließend werden daher Argumente gegenübergestellt, die für und gegen eine Realisierung mit der in dieser Arbeit gezeigten XML-basierten Architektur sprechen.

Vorteilhaft für eine Realisierung als XML-basierte Architektur sind folgende Aspekte:

- Die Anwendung arbeitet hauptsächlich mit strukturierten Dokumenten.
- In der Anwendungsdomäne werden XML Schemata gepflegt.
- Benutzer bearbeiten Daten in Formularform.
- Es werden Webservices eingebunden oder angeboten.
- Der Auftraggeber erwartet ein Datenformat, mit dem die weitere Verwendung der Daten einfach ist.

Ungünstig für den Einsatz der XML-basierten Architektur sind folgende Gegebenheiten:

- Die Anwendung arbeitet hauptsächlich mit unstrukturierten Daten, die sich nicht gut als XML-Dokumente repräsentieren lassen.
- Es kommen proprietäre Datenformate zum Einsatz, für die bereits Bibliotheken existieren.
- Die Anwendung baut als Erweiterung auf bestehenden Komponenten auf.
- Der Auftraggeber schreibt die Verwendung bestimmter Komponenten vor.

Fazit Die hier gezeigte XML-basierte Architektur stellt für viele Domänen eine leichtgewichtige Alternative zu traditionellen Entwicklungen dar. Im Rahmen des Single-Source-Projekts konnte durch den Einsatz der Architektur ein auf dem ODM-Standard basierendes System entwickelt und in Betrieb genommen werden. Weiterer Forschungsbedarf besteht in der Identifikation nötiger Maßnahmen, mit denen die XML-basierten Komponenten hinsichtlich ihrer Reife das Niveau von relationalen DBMS und objektorientierten Frameworks erreichen können. Die Erprobung der XML-basierten Architektur in weiteren Domänen, wie z. B. den in Abschnitt 6.3 skizzierten, steht ebenfalls noch aus.

A Anhang

A.1 XML Schema Beispiel

Programm A.1 zeigt die Implementierung des in Abschnitt 4.3.2 entworfenen XML Schemas, Programm A.2 zeigt ein dazu passendes Beispieldokument.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="Subject">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="AccessPermission" minOccurs="0" maxOccurs="
               unbounded">
7            <xs:complexType>
8              <xs:attribute name="UserOID" type="oid" use="required"/>
9            </xs:complexType>
10         </xs:element>
11         <xs:element name="DocumentationItem" minOccurs="0" maxOccurs="
               unbounded">
12           <xs:complexType>
13             <xs:sequence>
14               <xs:element name="URL" minOccurs="1" maxOccurs="1" type="
                     xs:anyURI">
15               </xs:element>
16               <xs:element name="Scheduled" minOccurs="1" maxOccurs="
                     unbounded" type="xs:date">
17               </xs:element>
18             </xs:sequence>
19             <xs:attribute name="Status" type="docstatus"/>
20           </xs:complexType>
21         </xs:element>
22       </xs:sequence>
23       <xs:attribute name="PatientKey" type="oid" use="required"/>
24       <xs:attribute name="SubjectKey" type="oid" use="required"/>
25       <xs:attribute name="StudyOID" type="oid" use="required"/>
26       <xs:attribute name="MetaDataVersionOID" type="oid" use="required"/>
27       <xs:attribute name="StudyName" type="xs:string"/>
28     </xs:complexType>
29   </xs:element>
30   <xs:simpleType name="oid">
31     <xs:restriction base="xs:string">
32       <xs:minLength value="1"/>
33     </xs:restriction>
34   </xs:simpleType>
35   <xs:simpleType name="docstatus">
36     <xs:restriction base="xs:string">
```

```
37          <xs:enumeration value="new"/>
38          <xs:enumeration value="started"/>
39          <xs:enumeration value="completed"/>
40        </xs:restriction>
41      </xs:simpleType>
42    </xs:schema>
```

Programm A.1: Umsetzung des Entwurfs in ein XSD-Schema.

```
1   <Subject PatientKey="UKM.0815" SubjectKey="doe" StudyOID="S.PRURITUS"
        MetaDataVersionOID="MD.1.0" StudyName="Juckreizstudie">
2     <AccessPermission UserOID="mustermann"/>
3     <AccessPermission UserOID="musterfrau"/>
4     <DocumentationItem Status="completed">
5       <URL>https://example.org/EDC…/1 </URL>
6       <Scheduled>2013-10-24</Scheduled>
7       <Scheduled>2013-10-31</Scheduled>
8     </DocumentationItem>
9     <DocumentationItem Status="new">
10      <URL>https://example.org/EDC…/2 </URL>
11      <Scheduled>2014-10-24</Scheduled>
12      <Scheduled>2014-10-31</Scheduled>
13    </DocumentationItem>
14  </Subject>
```

Programm A.2: Beispieldokument zu dem in Programm A.1 gezeigten XML Schema.

A.2 Entwurfsmuster Translator

Die in Abschnitt 4.4.4 beschriebene und hier gezeigte Implementierung des Entwurfsmusters Translator basiert auf den Code-Fragmenten aus [CBK10]. Programm A.3 zeigt die naive Implementierung, bei der Transformationsfunktionen für unterschiedliche Quelldokumente untereinander und mit Aspekten der Ablaufsteuerung vermischt werden. Programm A.4 zeigt den Aufruf dieses Ansatzes.

Sodann wird der Code durch Anwendung des Entwurfsmusters modularisiert: In Programm A.5 befindet sich die Ablaufsteuerung, die ohne Verweise zu den Templates auskommt. In Programm A.6 ist das Template zur Transformation der vCard-Elemente gezeigt, in Programm A.7 das Template zur Transformation der ODM-Elemente. Darin befinden sich nur die Transformationsregeln für die Elemente, es ist kein Code zur Ablaufsteuerung enthalten und die Templates weisen untereinander keine Abhängigkeiten auf. Programm A.8 zeigt den Aufruf der Transformationsfunktion mit Übergabe der Templatefunktionen als Parameter.

Quell- und Zieldokument für beide Ansätze befinden sich in Programm A.9 und Programm A.10.

```
1  xquery version ”3.0”;
2
3  module namespace templates=”http://uni-muenster.de/singlesource/all-
       templates”;
4
5  declare namespace vcard= ”urn:ietf:params:xml:ns:vcard-4.0”;
6  declare namespace odm=”http://www.cdisc.org/ns/odm/v1.3”;
7
8  declare function templates:get-user($doc){
9    for $user in ($doc/odm:AdminData/odm:User, $doc/vcard:vcards/vcard:vcard
         )
10   return
11   <User id=”{$user/@OID/string(),  $user/vcard:fn/vcard:text/string()}”>{
12     templates:get-surname($user),
13     templates:get-lastname($user),
14     templates:get-email($user)
15   }
16   </User>};
17
18 declare function templates:get-surname($user){
19   for $surname in ($user/odm:FirstName, $user/vcard:n/vcard:surname)
20   return
21 <Vorname>{
22   $surname/string()
23   }
24 </Vorname>};
25
26 declare function templates:get-lastname($user){
27   for $surname in ($user/odm:LastName, $user/vcard:n/vcard:given)
28   return
29 <Nachname>{
30   $surname/string()
31   }
32 </Nachname>};
33
34 declare function templates:get-email($user){
35   for $email in ($user/odm:Email, $user/vcard:email/vcard:text)
36   return
37 <EMail>{
38   $email/string()
39   }
40 </EMail>};
```

Programm A.3: Naive Implementierung eines Transformationsalgorithmus für Personendaten.

```
1  xquery version ”3.0”;
2
3  import module namespace templates=”http://uni-muenster.de/singlesource/all
       -templates” at ”everything.xql”;
4
5  <Nutzer>{
6    let $doc := doc(’addressdata.xml’)
7    return templates:get-user($doc/*)
8    }
```

```
9  </Nutzer>
```

Programm A.4: Aufruf der naiven Implementierung des Transformationsalgorithmus für Personendaten.

```
1  xquery version ”3.0”;
2
3  module namespace converter=”http://uni-muenster.de/singlesource”;
4
5  declare function converter:transform(
6    $i as item(),
7    $templates as function(*)+)
8    as item()*
9  {
10   for $tpl in $templates
11   let $template := $tpl()
12   let $match := $template[1] treat as function(*)
13   let $apply := $template[2] treat as function(*)
14   return
15     if($match($i))
16     then $apply($i, converter:transform#2, $templates)
17     else()
18 };
```

Programm A.5: Funktion `transform`, die die Anwendung der Templates steuert.

```
1  xquery version ”3.0”;
2
3  module namespace vcard-template=”http://uni-muenster.de/singlesource/vcard
       -templates”;
4  declare namespace vcard= ”urn:ietf:params:xml:ns:vcard-4.0”;
5
6  declare function vcard-template:match-vcards($item) as xs:boolean
7  {
8    $item instance of element(vcard:vcards)
9  };
10
11 declare function vcard-template:match-vcard($item) as xs:boolean
12 {
13   $item instance of element(vcard:vcard)
14 };
15
16 declare function vcard-template:match-name($item) as xs:boolean
17 {
18   $item instance of element(vcard:n)
19 };
20
21 declare function vcard-template:match-email($item) as xs:boolean
22 {
23   $item instance of element(vcard:email)
24 };
25
26 declare function vcard-template:apply-vcards(
27   $vcards as element(vcard:vcards),
28   $transform as (function(item()*, function(*)+) as item()*),
29   $templates as function(*)+)
```

```
30  {
31    for $child in $vcards/*
32    return $transform($child, $templates)
33  };
34
35  declare function vcard-template:apply-vcard(
36    $user as element(vcard:vcard),
37    $transform as (function(item()*, function(*)+) as item()*),
38    $templates as function(*)+)
39  {
40    <User id="{$user/vcard:fn/vcard:text/text()}">{
41       for $child in $user/*
42       return $transform($child, $templates)
43      }
44    </User>
45  };
46
47  declare function vcard-template:apply-name(
48    $name as element(vcard:n),
49    $transform as (function(item()*, function(*)+) as item()*),
50    $templates as function(*)+)
51  {
52    <Vorname>{
53      $name/vcard:surname/text()
54      }
55    </Vorname>,
56    <Nachname>{
57      $name/vcard:given/text()
58      }
59    </Nachname>
60  };
61
62  declare function vcard-template:apply-email(
63    $email as element(vcard:email),
64    $transform as (function(item()*, function(*)+) as item()*),
65    $templates as function(*)+)
66  {
67    <EMail>{
68      $email/vcard:text/text()
69      }
70    </EMail>
71  };
72
73  declare function vcard-template:get-templates(){
74    function () {vcard-template:match-vcards#1, vcard-template:apply-vcards
          #3},
75    function () {vcard-template:match-vcard#1, vcard-template:apply-vcard
          #3},
76    function () {vcard-template:match-name#1, vcard-template:apply-name#3},
77    function () {vcard-template:match-email#1, vcard-template:apply-email#3}
78    };
```

Programm A.6: Template für das Format vCard.

```
1  xquery version "3.0";
2
```

```
3  module namespace odm-template="http://uni-muenster.de/singlesource/odm-
       templates";
4  declare namespace odm="http://www.cdisc.org/ns/odm/v1.3";
5
6  declare function odm-template:match-admindata($item) as xs:boolean
7  {
8    $item instance of element(odm:AdminData)
9  };
10
11 declare function odm-template:match-user($item) as xs:boolean
12 {
13   $item instance of element(odm:User)
14 };
15
16 declare function odm-template:match-firstname($item) as xs:boolean
17 {
18   $item instance of element(odm:FirstName)
19 };
20
21 declare function odm-template:match-lastname($item) as xs:boolean
22 {
23   $item instance of element(odm:LastName)
24 };
25
26 declare function odm-template:match-email($item) as xs:boolean
27 {
28   $item instance of element(odm:Email)
29 };
30
31 declare function odm-template:apply-admindata(
32   $user as element(odm:AdminData),
33   $transform as (function(item()*, function(*)+) as item()*),
34   $templates as function(*)+)
35 {
36   for $child in $user/*
37   return $transform($child, $templates)
38 };
39
40 declare function odm-template:apply-user(
41   $user as element(odm:User),
42   $transform as (function(item()*, function(*)+) as item()*),
43   $templates as function(*)+)
44 {
45   <User id="{$user/@OID}">{
46      for $child in $user/*
47      return $transform($child, $templates)
48     }
49   </User>
50 };
51
52 declare function odm-template:apply-firstname(
53   $name as element(odm:FirstName),
54   $transform as (function(item()*, function(*)+) as item()*),
55   $templates as function(*)+)
56 {
57   <Vorname>{
58     $name/text()
```

```
59      }
60    </Vorname>
61  };
62
63  declare function odm-template:apply-lastname(
64    $name as element(odm:LastName),
65    $transform as (function(item()*, function(*)+) as item()*),
66    $templates as function(*)+)
67  {
68    <Nachname>{
69      $name/text()
70      }
71    </Nachname>
72  };
73
74  declare function odm-template:apply-email(
75    $email as element(odm:Email),
76    $transform as (function(item()*, function(*)+) as item()*),
77    $templates as function(*)+)
78  {
79    <EMail>{
80      $email/text()
81      }
82    </EMail>
83  };
84
85  declare function odm-template:get-templates(){
86    function () {odm-template:match-admindata#1, odm-template:apply-
          admindata#3},
87    function () {odm-template:match-user#1, odm-template:apply-user#3},
88    function () {odm-template:match-firstname#1, odm-template:apply-
          firstname#3},
89    function () {odm-template:match-lastname#1, odm-template:apply-lastname
          #3},
90    function () {odm-template:match-email#1, odm-template:apply-email#3}
91    };
```

Programm A.7: Template für das Format ODM.

```
1  xquery version ”3.0”;
2
3  import module namespace converter=”http://uni-muenster.de/singlesource” at
       ”translator.xql”;
4  import module namespace odm-template=”http://uni-muenster.de/singlesource/
       odm-templates” at ”odm-template.xql”;
5  import module namespace vcard-template=”http://uni-muenster.de/
       singlesource/vcard-templates” at ”vcard-template.xql”;
6
7  <Nutzer>{
8    let $doc := doc(’addressdata.xml’)
9    for $user in ($doc/mixedData/*)
10   return converter:transform($user, (odm-template:get-templates(), vcard-
         template:get-templates()))
11 }
```

```
12  </Nutzer>
```

Programm A.8: Anwendung des Entwurfsmusters Translator: Aufruf der Funktion `transform`.

```
1   <mixedData>
2     <AdminData xmlns="http://www.cdisc.org/ns/odm/v1.3">
3       <User OID="forster" UserType="Other">
4         <FirstName>Christian</FirstName>
5         <LastName>Forster</LastName>
6         <Email>christian.forster@ercis.de</Email>
7       </User>
8       <User OID="musterfrau" UserType="Investigator">
9         <FirstName>Petra</FirstName>
10        <LastName>Musterfrau</LastName>
11        <Email>petra@example.org</Email>
12      </User>
13      <User OID="mustermann" UserType="Investigator">
14        <FirstName>Max</FirstName>
15        <LastName>Mustermann</LastName>
16        <Email>max@example.org</Email>
17        <Email>maximilian@example.org</Email>
18      </User>
19    </AdminData>
20    <vcards xmlns="urn:ietf:params:xml:ns:vcard-4.0">
21      <vcard>
22        <fn>
23          <text>Emine Kartal</text>
24        </fn>
25        <n>
26          <surname>Emine</surname>
27          <given>Kartal</given>
28          <additional/>
29          <prefix/>
30          <suffix/>
31        </n>
32        <email>
33          <parameters>
34            <type>
35              <text>home</text>
36            </type>
37          </parameters>
38          <text>emine@example.org</text>
39        </email>
40      </vcard>
41      <vcard>
42        <fn>
43          <text>Franz-Xaver Gabler</text>
44        </fn>
45        <n>
46          <surname>Franz-Xaver</surname>
47          <given>Gabler</given>
48          <additional/>
49          <prefix/>
50          <suffix/>
51        </n>
```

```
52        <email>
53          <parameters>
54            <type>
55              <text>home</text>
56            </type>
57          </parameters>
58          <text>franz-xaver@example.org</text>
59        </email>
60        <email>
61          <parameters>
62            <type>
63              <text>work</text>
64            </type>
65          </parameters>
66          <text>gabler@example.org</text>
67        </email>
68      </vcard>
69    </vcards>
70  </mixedData>
```

Programm A.9: Quelldokument `addressdata.xml`, das Addressdaten in den Formaten ODM und vCard enthält.

```
1   <Nutzer>
2     <User id="forster">
3       <Vorname>Christian</Vorname>
4       <Nachname>Forster</Nachname>
5       <EMail>christian.forster@ercis.de</EMail>
6     </User>
7     <User id="musterfrau">
8       <Vorname>Petra</Vorname>
9       <Nachname>Musterfrau</Nachname>
10      <EMail>petra@example.org</EMail>
11    </User>
12    <User id="mustermann">
13    <Vorname>Max</Vorname>
14      <Nachname>Mustermann</Nachname>
15      <EMail>max@example.org</EMail>
16      <EMail>maximilian@example.org</EMail>
17    </User>
18    <User id="Emine Kartal">
19      <Vorname>Emine</Vorname>
20      <Nachname>Kartal</Nachname>
21      <EMail>emine@example.org</EMail>
22    </User>
23    <User id="Franz-Xaver Gabler">
24      <Vorname>Franz-Xaver</Vorname>
25      <Nachname>Gabler</Nachname>
26      <EMail>franz-xaver@example.org</EMail>
27      <EMail>gabler@example.org</EMail>
28    </User>
29  </Nutzer>
```

Programm A.10: Ergebnis der Anwendung des Entwurfsmusters Translator auf die Datei `addressdata.xml` aus Programm A.9.

A.3 Implementierungsbeispiele des Gerüsts

A.3.1 MVC

Dieser Abschnitt zeigt die Implementierung der in Abschnitt 5.1.1 entwickelten Komponenten des MVC-Musters.

Implementierung Controller Programm A.11 zeigt das Gerüst des Controllers, der als Hauptmodul ein XML-Element zur Steuerung des Kontrollflusses zurückgibt. Lokale Funktionsdefinitionen finden sich im Bereich bis Zeile 54. Zunächst wird in Zeile 12 eine Funktion `redirect`, die das XML-Fragment für Umleitungen von Anfragen konstruiert, deklariert. Als Parameter werden die relative Ziel-URL und ggf. anzuhängende URL-Parameter übergeben. Die Funktion `require-login` in Zeile 23 prüft, ob der Nutzer in der Session angemeldet ist, oder ob er zur Login-Seite weitergeleitet werden muss. Für den letzten Fall wird das entsprechende XML-Fragment generiert, das die ursprünglich angefragte URL als Parameter anhängt, um sie nach dem Login wieder aufrufen zu können. Die zur Prüfung der Session benötigte Funktion `auth:is-logged-in` stammt aus der Bibliothek für das Rechtemanagement, das in einem folgenden Abschnitt erläutert wird. Die Funktion `require-access` in Zeile 39 führt die Überprüfung, ob ein Nutzer ausreichende Rechte für die angefragte Operation hat, durch. Als Parameter werden eine Zeichenkette zur Identifikation der angefragten Operation und die vom Controller zur Weiterleitung vorgesehene URL übergeben. Zunächst wird sichergestellt, dass ein Nutzer eingeloggt ist. Dann wird geprüft, ob dieser Benutzer die entsprechende Berechtigung hat, wozu die Funktion `abac:request-access` verwendet wird. Dieser werden der angemeldete Benutzer und die Identifikation für die angefragte Operation übergeben. Ergibt die Prüfung, dass der Benutzer berechtigt ist, wird ein XML-Fragment zur Weiterleitung auf die vorgesehene View zurückgegeben. Andernfalls wird eine Umleitung auf die Startseite der Anwendung mit einem Parameter `error`, der die Fehlerursache angibt, vorgenommen. Nach diesen Funktionsdeklarationen folgen die eigentlichen Routing-Anweisungen in Form von if-then-else-Konstrukten. Sobald ein passendes Ziel gefunden wurde, wird die Funktion `require-access` mit den entsprechenden Parametern aufgerufen, damit das entsprechende XML-Element generiert und zurückgegeben wird. Daher ist die Reihenfolge der Anweisungen entscheidend. Die Behandlung der Ziele `login` (Zeile 65) und `logout` (Zeile 79) nimmt hierbei eine Sonderstellung ein. Beim Aufruf der Login-Seite erfolgt eine Weiterleitung auf die Startseite der Webanwendung, wenn der Nutzer bereits eingeloggt ist. Somit

kann die Login-Seite als Lesezeichen im Browser angelegt werden und bei bereits bestehender Session werden die Login-Daten nicht erneut abgefragt. Falls die Session, z. B. durch Inaktivität, abgelaufen ist, wurde der Nutzer von seiner eigentlichen Anfrage auf die Login-Seite umgeleitet. Damit nach dem Login die eigentliche Anfrage ausgeführt wird, wird der Parameter `redirect_uri` ausgewertet und falls vorhanden der Nutzer auf die angegebene Seite umgeleitet. Andernfalls hatte der Login seinen Ursprung in der Login-Seite und der Nutzer wird auf die Startseite weitergeleitet. Das Ausloggen ist stets möglich und benötigt nur den Aufruf der bereits erwähnten Funktion `logout`. Weitere anwendungsspezifische Weiterleitungsmuster werden im Bereich um Zeile 82 hinzugefügt. Sie bilden den Kern der Routing-Logik und werden für jede vorgesehene URL angelegt. Falls eine Anfrage zu keinem Muster passt, fängt die letzte `else`-Anweisung in Zeile 87 die Anfrage auf und löst eine Weiterleitung auf die Startseite mit entsprechendem Parameter für die Angabe der Fehlerursache aus.

```
1  xquery version "1.0";
2
3  import module namespace auth = "app.auth";
4  import module namespace abac = "app.abac";
5
6  (:~
7   : Constructs a local redirection XML fragment.
8   : @param $path the local target of the redirection
9   : @param $params parameter to add to the request URL
10  : @return exist-specific routing information
11  :)
12 declare function local:redirect($path as xs:string, $params as element()*)
       as element(){
13   <dispatch xmlns="http://exist.sourceforge.net/NS/exist">
14     <redirect url="/exist/{$exist:controller}/{$path}{if ($params) then
           '?' else ()}{ for $param in $params/add-parameter return (concat
           ($param/@name,'=',encode-for-uri($param/@value),'&amp;'))}">
15     </redirect>
16   </dispatch>
17 };
18
19 (:~
20  : Ensure that a user is logged in.
21  : @return empty, if user is logged in. XML fragment for redirection to
        login page, else.
22  :)
23 declare function local:require-login() as element()?{
24   let $logged-in := auth:is-logged-in()
25   return
26   if($logged-in) then ()
27   else
28     local:redirect('login',
29       <params xmlns="http://exist.sourceforge.net/NS/exist">
30         <add-parameter name="redirect_uri" value="{concat(request:get-uri
              (),'?',request:get-query-string())}"/>
31       </params>)
```

```
32  };
33
34  (:~
35   : Forward to a requested resource, if access is granted. Redirect to
36   : home and set error, else.
37   : @return forward URL, if user is logged in. XML fragment for redirection
          to home + error, else.
38   :)
39  declare function local:require-access($resource-id as xs:string, $
         forward_uri as xs:anyURI) as element(){
40    (: login is required :)
41    let $l := local:require-login()
42    return
43      if ($l) then $l (: redirect to login :) else (: user already there :)
44      if(abac:request-access(auth:get-user-node(),(),$resource-id)/name() eq
             'granted')
45      then (: access granted :)
46        <dispatch xmlns="http://exist.sourceforge.net/NS/exist">
47          <forward url="/{$exist:controller}{$forward_uri}"/>
48        </dispatch>
49      else (: Access denied :)
50        local:redirect('home',
51          <params xmlns="http://exist.sourceforge.net/NS/exist">
52            <add-parameter name="error" value="accessdenied"/>
53          </params>)
54  };
55
56  (:~
57   : Calls the logout function and redirects to login
58   :)
59  declare function local:logout() as element(){
60    auth:logout(),
61    local:redirect('login',())
62  };
63
64  (: Forwards and Redirects :)
65  if ($exist:path eq '/login') then
66    if (auth:is-logged-in())
67    then local:redirect('home', ())
68    else
69    let $login := auth:login-user()
70    return
71      if ($login) then (: if redirection is set, then redirect to the target
             , else to home :)
72        <dispatch xmlns="http://exist.sourceforge.net/NS/exist">
73          <redirect url="{request:get-parameter('redirect_uri', concat('/
                exist',$exist:controller,'/home'))}" />
74        </dispatch>
75      else (: in case of login failure, login again :)
76        <dispatch xmlns="http://exist.sourceforge.net/NS/exist">
77          <forward url="/{$exist:controller}/view/login.xql" />
78        </dispatch>
79  else if ($exist:path eq '/logout') then
80    local:logout()
81
82  else if ($exist:path eq '/home') then (: go to application startpage :)
83    local:require-access('url-home',xs:anyURI( '/view/home.xql'))
```

```
84  else if ($exist:path eq '/data/patient') then
85    local:require-access('url-patientshow', xs:anyURI( '/view/show_form.xql'
          ))
86    (: add more routing here :)
87  else (: go to home if nothing else matches :)
88    local:redirect('home',
89      <params xmlns="http://exist.sourceforge.net/NS/exist">
90        <add-parameter name="error" value="resourcenotfound"/>
91      </params>)
```

Programm A.11: Implementierung des Controllers.

Implementierung View Die Implementierung einer View ist in Programm A.12 gezeigt. Es handelt sich dabei um ein Gerüst, das je nach Zweck ausgebaut wird. Zuerst erfolgen die Modulimporte. Die Funktionen aus dem Bibliotheksmodul `app-html` liefern HTML-Fragmente, die in mehreren Seiten benötigt werden und daher ausgelagert sind. Zum Teil geben sie feste Ausdrücke zurück, z. B. `htmlfooter()` für die Fußzeile, es sind auch solche mit Parameterübergabe möglich, z. B. `format-headerline($arg1, $arg2)` für Überschrift und Untertitel der Seite. In Abhängigkeit von externen Konfigurationsparametern integriert `tracker()` JavaScript-Code für ein Analysewerkzeug. Das Modul `app-logic` steht exemplarisch für die Einbindung von Model-Bibliotheken. Sodann werden in Zeile 7 die Parameter des HTTP-Requests ausgelesen und an Variablen gebunden. Diese können von der View an die Geschäftslogik weitergegeben oder innerhalb der View verwendet werden. In Zeile 9 wird eine Funktion des Models verwendet, um den Inhalt der Seite zu generieren. So kann z. B. dynamisch ein XForms-Formular erzeugt werden. Ab Zeile 12 wird das HTML-Gerüst erzeugt. Die in geschweiften Klammern angegebenen XQuery-Ausdrücke werden interpretiert und durch ihre Werte ersetzt.

```
1  xquery version "1.0";
2
3  import module namespace appUtil = "app";
4  import module namespace app-html = "app.html";
5  import module namespace app-logic = "app.logic";
6
7  let $user-name := request:get-parameter("user_name", "")
8  let $request-parameter := request:get-parameter("parameter", "")
9  let $content := app-logic:generate-form($request-parameter)
10
11 return
12 <html>
13   { app-html:htmlheader() }
14   <body>
15     <div class="container">
16       <div class="content">
17         <div class="page-header">
18           {app-html:headerright()}
```

```
19            {app-html:format-headline("Titel der Aktivität", ())}
20          </div>
21          { app-html:headerline("Daten", "", "Formulare", concat($
                appUtil:rootURL, "/data/list?parameter_=", $request_parameter
                ), concat("Formular-ID: ", $request-parameter)) }
22          {
23          if($content/name() eq "error") then
24              app-html:alert("Fehler", $content, "error")
25          else $content
26          }
27        </div>
28        { app-html:htmlfooter() }
29      </div>
30      { app-html:tracker() }
31    </body>
32  </html>
```

Programm A.12: Implementierung eines Gerüstes für eine View-Komponente.

Implementierung Model Die Bibliothek `app.logic` in Programm A.13 zeigt die in Programm A.12 verwendete Komponente des Models. Die in Zeile 9 befindliche Funktion `generate-form` zeigt beispielhaft den Aufruf einer Transformation für ein Datenelement. Zunächst wird das Datenelement aus dem Domänenmodell mit der Funktion `get-data` geladen und ein Verweis auf die XSLT-Anweisung wird an die Variable `$stylesheet` gebunden. Dann wird die Transformation mit der eXist-db-spezifischen Funktion `transform` aus dem Namensraum `transform` durchgeführt. Nach einer hier nicht näher ausgeführten Überprüfung des Resultats in Zeile 15 erfolgt entweder die Rückgabe des Formulars oder eines `error`-Elements.

```
1   module namespace app-logic = "app.logic";
2
3   import module namespace app-model-domain = "app.model.domain";
4
5   (:~ generate a form of the given data element
6   : @param $parameter identites the data element
7   : @return the generated form
8   :)
9   declare function app-logic:generate-form($get-data($parameter)){
10    let $data := app-model-domain:get-data($parameter)
11    let $stylesheet := xs:anyURI("xslt/formgeneration.xsl")
12
13    let $form := transform:transform($data, $stylesheet)
14    return
15    if (app-logic:check-validity($form))
16    then $form
17    else <error>invalid</error>
18  };
```

Programm A.13: Implementierung eines Gerüstes für eine Model-Komponente.

A.3.2 XML Schema und Implementierung der Nutzerverwaltung

Dieser Abschnitt zeigt das in Abschnitt 5.1.2 entwickelte Datenmodell.

Das XML Schema `users.xsd` befindet sich in Programm A.14. Von Zeile 6 bis Zeile 12 wird die Definition der Gruppen angelegt. Ab Zeile 59 erfolgt die Typdefinition des dazu verwendeten Typs `groupType`. Der für das Attribut `name` verwendete Datentyp xs:ID stellt sicher, dass das Attribut Schlüsseleigenschaften aufweist. Zwischen Zeile 13 und Zeile 54 wird das Element `users` angelegt, das die einzelnen Elemente `user` enthält. Die Gruppenbezeichnungen sind dazu der Domäne der klinischen Studien entnommen. Durch die Verwendung des Datentyps `xs:IDREF` in Zeile 33 für das Attribut `name` wird sichergestellt, dass es sich beim Verweis auf die Gruppe um einen gültigen Fremdschlüssel handelt, also nur auf tatsächlich existierende Gruppen referenziert wird. Der Datentyp `xs:ID` ist ein globaler Schlüssel, d. h. jeder Attributwert kann nur einmal vergeben werden, unabhängig vom Element, das das Attribut enthält. Für das Element `attribute` in Zeile 19 muss daher ein anderes Verfahren gewählt werden, um einen Schlüssel zu bestimmen, da innerhalb unterschiedlicher Gruppen Attribute mit demselben Namen auftreten dürfen. Die Festlegung von `name` als Schlüssel in Zeile 35 sorgt mit den Elementen `selector` und `path` dafür, dass alle Elemente `attribute` im Kontext einer Gruppe unterschiedliche Attribute beschreiben und somit keine doppelten Attributbestimmungen erfolgen können.

Programm A.15 zeigt zur Veranschaulichung ein Beispieldokument, das dem Schema folgt.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="auth">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="userGroups">
7            <xs:complexType>
8              <xs:sequence>
9                <xs:element name="groupDefinition" type="groupType"
                      minOccurs="0" maxOccurs="unbounded"/>
10             </xs:sequence>
11           </xs:complexType>
12         </xs:element>
13         <xs:element name="users" minOccurs="1" maxOccurs="1">
14           <xs:complexType>
15             <xs:sequence>
16               <xs:element name="user" minOccurs="1" maxOccurs="unbounded">
17                 <xs:complexType>
18                   <xs:sequence>
19                     <xs:element name="group" minOccurs="1" maxOccurs="
                            unbounded">
20                       <xs:complexType>
21                         <xs:sequence>
```

```
22                          <xs:element name="attribute" minOccurs="0"
                                 maxOccurs="unbounded">
23                            <xs:complexType>
24                              <xs:simpleContent>
25                                <xs:extension base="xs:string">
26                                  <xs:attribute name="name"/>
27                                  <xs:attribute name="value"/>
28                                </xs:extension>
29                              </xs:simpleContent>
30                            </xs:complexType>
31                          </xs:element>
32                        </xs:sequence>
33                        <xs:attribute name="name" type="xs:IDREF" use="
                              required"/>
34                      </xs:complexType>
35                      <xs:key name="attributekey">
36                        <xs:selector xpath="./attribute"/>
37                        <xs:field xpath="@name"/>
38                      </xs:key>
39                    </xs:element>
40                  </xs:sequence>
41                  <xs:attribute name="name" type="xs:string"/>
42                  <xs:attribute name="password"/>
43                  <xs:attribute name="activated"/>
44                  <xs:attribute name="salt"/>
45                  <xs:attribute name="method"/>
46                </xs:complexType>
47              </xs:element>
48            </xs:sequence>
49          </xs:complexType>
50          <xs:key name="userkey">
51            <xs:selector xpath="./user"/>
52            <xs:field xpath="@name"/>
53          </xs:key>
54        </xs:element>
55      </xs:sequence>
56      <xs:attribute name="version"/>
57    </xs:complexType>
58  </xs:element>
59  <xs:complexType name="groupType">
60    <xs:sequence>
61      <xs:element name="requiredAttribute" minOccurs="0" maxOccurs="
            unbounded" type="xs:string"/>
62      <xs:element name="groupDefinition" type="groupType" minOccurs="0"
            maxOccurs="unbounded"/>
63    </xs:sequence>
64    <xs:attribute name="name" type="xs:ID" use="required"/>
65    <xs:attribute name="displayName"/>
66  </xs:complexType>
67 </xs:schema>
```

Programm A.14: Das XML-Schema `users.xsd` der Nutzerverwaltung.

```
1  <auth version="1.0">
2    <userGroups>
3      <groupDefinition Name="admin" DisplayName="Administrator">
4        <groupDefinition Name="pi" DisplayName="Hauptauswerter">
```

```
5           <requiredAttribute>StudyOID</requiredAttribute>
6           <groupDefinition Name="siteadmin" DisplayName="
                Standortadministrator">
7             <requiredAttribute>StudyOID</requiredAttribute>
8             <groupDefinition Name="investigator" DisplayName="Auswerter">
9               <requiredAttribute>StudyOID</requiredAttribute>
10              <requiredAttribute>LocationOID</requiredAttribute>
11              <groupDefinition Name="dataaggregator" DisplayName="
                    Datensammler">
12                <requiredAttribute>StudyOID</requiredAttribute>
13                <requiredAttribute>LocationOID</requiredAttribute>
14              </groupDefinition>
15            </groupDefinition>
16          </groupDefinition>
17        </groupDefinition>
18      </groupDefinition>
19    </userGroups>
20    <users>
21      <user name="forster" password="8
            a0bc95e4b77724742ccecc3afa81fdfdf571d6656d485e4a27c65393f3d29c"
            activated="true" salt="0.8158123765339833" method="sha-256">
22        <group name="admin">
23        </group>
24      </user>
25      <user name="musterfrau" password="
            eb8024c139325acb6c346c9584cb86890851a5e874e9b2736d693aa5d374b964"
              activated="true" salt="0.42581624484256375" method="sha-256">
26        <group name="pi">
27          <attribute name="StudyOID" value="beispielstudie"/>
28        </group>
29      </user>
30      <user name="mustermann" password="
            fd6800e133e71e541fad14f07f404adad450fabfc3e6102a7c0eb43163d0cc36"
              activated="true" salt="0.7847878953888499" method="sha-256">
31        <group name="dataaggregator">
32          <attribute name="StudyOID" value="beispielstudie"/>
33          <attribute name="LocationOID" value="UKM"/>
34        </group>
35      </user>
36    </users>
37  </auth>
```

Programm A.15: Beispieldokument, das konform zum XML-Schema `users.xsd` der Nutzerverwaltung ist.

A.3.3 Implementierung Rechtemanagement

Dieser Abschnitt zeigt die Implementierung der in Abschnitt 5.1.3 entworfenen Funktionalität zum Rechtemanagement.

Authentifizierung Die Bibliothek zur Authentifizierung der Nutzer mit dem Namensraum `app.auth` verwaltet die Sessions und implementiert Funktionen,

um Nutzer zu identifizieren und durch Hinterlegen des Attributs `app.user` in der Session an die Webanwendung anzumelden. Die Abmeldung durch Timeout ist eine Funktion der zugrundeliegenden Sessionverwaltung des Application Servers und wird in diesem konfiguriert. Für die Implementierung der Bibliothek werden in Zeile 5 f. zwei Module eingebunden: Das Modul mit dem Präfix `app-user-model` stellt den Zugriff auf die Daten der im vorherigen Abschnitt gezeigten Nutzerverwaltung bereit. Das Modul mit dem Präfix `appUtil` stellt anwendungsspezifische Konfigurationsparameter zur Verfügung, z. B. DBMS-Login-Daten für die Webanwendung. Von den folgend bereitgestellten Funktionen entsprechen `login-user` (Zeile 23) und `logout` (Zeile 58) den Aktivitäten Anmelden und Abmelden aus dem Entwurf. Die Funktionen ab Zeile 86 dienen der Aktivität Anfrage zur Nutzeridentität. Die übrigen Funktionen erfüllen Aufgaben zur Unterstützung der genannten Funktionen, so führt die Funktion `authenticate-user` in Zeile 69 die Prüfung von Nutzernamen und Passwort mit durch die Nutzerverwaltung gespeicherten Daten inkl. Hashingmethode und Salt durch.

```
1  xquery version "1.0";
2
3  module namespace app-auth = 'app.auth';
4
5  import module namespace app-user-model = "app.model.user";
6  import module namespace appUtil = "app.utils";
7
8  (:~
9   : Store user credentials in session for future use.
10  : @param $user set this UserOID in session
11  : @return always true
12  :)
13 declare function app-auth:set-credentials($user as xs:string) as
       xs:boolean {
14   let $tmp := session:set-attribute("app.user", $user)
15   return
16     fn:true()
17 };
18
19 (:~
20  : Check if login parameters 'user' and 'password' are passed in the
       request. If yes, try to authenticate the user and store credentials
       into the session.
21  : @return indicating authentication success.
22  :)
23 declare function app-auth:login-user() as xs:boolean {
24   (: logout previous user to be on the save side :)
25   app-auth:logout(),
26   let $user := request:get-parameter("user", ())
27   let $password := request:get-parameter("password", ())
28   return
29     if (empty($user) or empty($password)) then fn:false()
30     else
```

```
31        let $tmp := session:create()
32        let $db-login := local:db-login()
33        return
34        if (app-auth:authenticate-user($user, $password))
35          then
36          (: user is written to session, put login hooks here :)
37          (
38          let $user-name := app-user-model:get-user($user)/@name
39          let $tmp := app-auth:set-credentials($user-name)
40          return $db-login
41          )
42        else
43          let $tmp := session:invalidate()
44          return fn:false()
45  };
46
47  (:~
48   : Login with database-level credentials
49   : @return true, if successfully authenticated
50   :)
51  declare function local:db-login() as xs:boolean {
52    xmldb:login($appUtil:dbpath, $appUtil:dbuser, $appUtil:dbpassword)
53  };
54
55  (:~
56   : User logout and invalidation of session
57   :)
58  declare function app-auth:logout() {
59    (: put logout hooks here :)
60    session:invalidate()
61  };
62
63  (:~
64   : Authenticates user with user in database
65   : @param $user username
66   : @param $password password
67   : @return true, if authenticated successfully
68   :)
69  declare function app-auth:authenticate-user($user as xs:string, $password
        as xs:string) as xs:boolean
70  {
71    let $user-node := app-user-model:get-user($user)
72    return
73    if(empty($user-node)) then fn:false()
74    else
75      let $pass := string($user-node/attribute::password),
76        $activated := string($user-node/attribute::activated),
77        $salted-password := concat($password, $user-node/@salt),
78        $hashed-password := util:hash($salted-password, $user-node/@method)
79      return
80      if ($pass != "" and $activated eq "true") then (
81        $hashed-password eq $pass
82        )
83      else fn:false()
84  };
85
86
```

```
87  (:~
88   : Check if there is a user logged in, no matter who
89   : @return true, if there is a user in the session
90   :)
91  declare function app-auth:is-logged-in() as xs:boolean {
92      exists(app-auth:credentials-from-session())
93  };
94
95  (:~
96   : @return oid of current user in the session
97   :)
98  declare function app-auth:get-user-name() as xs:string?
99  {
100   session:get-attribute("app.user")
101 };
102
103 (:~
104  : Receive user model from user in session
105  : @return user node
106  :)
107 declare function app-auth:get-user-node() as element()
108 {
109   let $user-oid := app-auth:get-user-name()
110   return app-user-model:get-user($user-oid)
111 };
```

Programm A.16: Implementierung der Authentifizierungsbibliothek.

Autorisierung Das XML Schema der Datenstruktur zur attributbasierten Berechtigungsprüfung ist in Programm A.17 wiedergegeben. Das Element `abac` fungiert als Wurzelelement. Darin befinden sich die Elemente `resources` (Zeile 5), `policies` (Zeile 17) und `clauses` (Zeile 35), die jeweils als übergeordnete Elemente für die einzelnen Elemente `resource`, `policy` und `clause` dienen. In Zeile 65 werden für diese Elemente die Schlüsselattribute und Fremdschlüsselbeziehungen angegeben. Ein dazu konformes, kurzes Regelwerk ist in Programm A.18 wiedergegeben. Es zeigt die Flexibilität der hinterlegbaren Regeln. In Zeile 21 ist eine stets wahre Regel hinterlegt, die Verwendung findet, wenn alle Nutzer auf eine Ressource zugreifen sollen. Sie wird z. B. beim Zugriff auf die Startseite der Anwendung benutzt. Die Regel `user_is_admin` in Zeile 30 fordert, dass ein Nutzer der Gruppe Administrator angehört. Da dazu keine Attribute benötigt werden, sind keine weiteren Überprüfungen erforderlich. Die Regel `user_is_dataaggregator_or_higher` in Zeile 39 ist am komplexesten. Sie prüft die im Request übergebenen Parameter und nutzt dazu eine Funktion `is-in-user-group-or-higher`, um zu prüfen, ob der den Request ausführende Nutzer mindestens der benötigten Gruppe Datensammler der angefragten Studie und des angefragten Standorts angehört. Der dritte der Funktion zu übergebende Parameter ist ein XML-Element, dessen Klammern <

und > durch < und > maskiert werden müssen, da das Regelwerk selbst im XML-Format gespeichert ist.

Bei der Implementierung der Anwendung müssen für jeden zu prüfenden Fall Regeln bestimmt werden. Diese sind nicht nur vom Controller für das Routing einsetzbar, sondern können auch durch die Geschäftslogik verwenden werden, um bspw. Filterfunktionen zu implementieren.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="abac">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="resources">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="resource" minOccurs="0" maxOccurs="
                   unbounded">
                <xs:complexType>
                  <xs:attribute name="id" type="xs:string"/>
                  <xs:attribute name="policy-id" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="policies">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="policy" minOccurs="0" maxOccurs="unbounded
                   ">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="clause" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:attribute name="id" type="xs:string"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="clauses">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="clause" minOccurs="0" maxOccurs="unbounded
                   ">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="condition" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="left-attribute" type="
                               attributeType"/>
```

```
45                          <xs:element name="op">
46                            <xs:complexType>
47                              <xs:attribute name="type" type="op-type"/>
48                            </xs:complexType>
49                          </xs:element>
50                          <xs:element name="right-attribute" type="
                                attributeType"/>
51                        </xs:sequence>
52                        <xs:attribute name="id" type="xs:string"/>
53                      </xs:complexType>
54                    </xs:element>
55                  </xs:sequence>
56                  <xs:attribute name="id" type="xs:string"/>
57                </xs:complexType>
58              </xs:element>
59            </xs:sequence>
60          </xs:complexType>
61        </xs:element>
62      </xs:sequence>
63      <xs:attribute name="version"/>
64    </xs:complexType>
65    <xs:key name="clausekey">
66      <xs:selector xpath="./clauses/clause"/>
67      <xs:field xpath="@id"/>
68    </xs:key>
69    <xs:key name="policykey">
70      <xs:selector xpath="./policies/policy"/>
71      <xs:field xpath="@id"/>
72    </xs:key>
73    <xs:key name="resourcekey">
74      <xs:selector xpath="./resources/resource"/>
75      <xs:field xpath="@id"/>
76    </xs:key>
77    <xs:keyref name="clauseref" refer="clausekey">
78      <xs:selector xpath="./policies/policy/clause"/>
79      <xs:field xpath="@id"/>
80    </xs:keyref>
81    <xs:keyref name="policyref" refer="policykey">
82      <xs:selector xpath="./resources/resource"/>
83      <xs:field xpath="@policy-id"/>
84    </xs:keyref>
85  </xs:element>
86  <xs:complexType name="attributeType">
87    <xs:simpleContent>
88      <xs:extension base="xs:string">
89        <xs:attribute name="owner" type="owner-type"/>
90      </xs:extension>
91    </xs:simpleContent>
92  </xs:complexType>
93  <xs:simpleType name="owner-type">
94    <xs:restriction base="xs:string">
95      <xs:enumeration value="resource"/>
96      <xs:enumeration value="value"/>
97      <xs:enumeration value="entity"/>
98    </xs:restriction>
99  </xs:simpleType>
100 <xs:simpleType name="op-type">
```

```
101       <xs:restriction base="xs:string">
102         <xs:enumeration value="=="/>
103         <xs:enumeration value="!="/>
104         <xs:enumeration value="in"/>
105       </xs:restriction>
106     </xs:simpleType>
107   </xs:schema>
```

Programm A.17: XML Schema `abac.xsd` der Datenstruktur zur attributbasierten Berechtigungsprüfung.

```
1   <abac>
2     <resources>
3       <!-- Access to URLs -->
4       <resource id="url-home" policy-id="free-for-all"/>
5       <resource id="url-addstudy" policy-id="url_access_admin"/>
6       <resource id="url-patientshow" policy-id="url_access_own_patient"/>
7     </resources>
8     <policies>
9       <policy id="free-for-all">
10        <clause id="tautology"/>
11      </policy>
12      <policy id="url_access_admin">
13        <clause id="user_is_admin"/>
14      </policy>
15      <policy id="url_access_own_patient">
16        <clause id="user_is_dataaggregator_or_higher"/>
17      </policy>
18    </policies>
19
20    <clauses>
21      <clause id="tautology">
22        <condition>
23          <left-attribute owner="value">true</left-attribute>
24          <op type="=="/>
25          <right-attribute owner="value">true</right-attribute>
26        </condition>
27      </clause>
28
29      <!-- this clause is true if entity is user and admin -->
30      <clause id="user_is_admin">
31        <condition>
32          <left-attribute owner="value">admin</left-attribute>
33          <op type="in"/>
34          <right-attribute owner="entity">group/@name</right-attribute>
35        </condition>
36      </clause>
37
38      <!--this clause is true if the user is dataaggregator or upwards for
            study_oid, metadataversion_oid and patient_oid  in request
            parameter -->
39      <clause id="user_is_dataaggregator_or_higher">
40        <condition>
41          <left-attribute owner="entity">
42            pruritus-abac:is-in-group-or-higher(@name, 'dataaggregator',
43              &lt;attributes&gt;
```

```
44              &lt;attribute name="StudyOID" value="{request:get-parameter('
                     study_oid', '')}"/&gt;
45              &lt;attribute name="LocationOID" value="{x4T-patient-model:get
                     -site(request:get-parameter('study_oid', ''), request:get
                     -parameter('metadataversion_oid', ''), request:get-
                     parameter('patient_oid',''))}"/&gt;
46            &lt;/attributes&gt;
47            )
48          </left-attribute>
49          <op type="=="/>
50          <right-attribute owner="value">true</right-attribute>
51        </condition>
52      </clause>
53    </clauses>
54  </abac>
```

Programm A.18: ABAC-Regelwerk, konform zum XML Schema `abac.xsd` aus Programm A.17.

A.3.4 Implementierung Webservices

Der Abschnitt zeigt die Implementierung der in Abschnitt 5.1.4 entworfenen Funktionalität zur Kommunikation mit Webservices. Das Modul mit dem Namensraum `app-ws` aus Programm A.19 implementiert die Funktionen für die Kommunikation mit einem SOAP-basierten Webservice. Durch die Komponenten der Webanwendung werden die Template-erzeugenden Funktionen, von denen hier exemplarisch die Funktion `prepopWS` (Zeile 10), und die Funktion `callService` (Zeile 22) zum Absenden einer Anfrage verwendet. Die Template-erzeugenden Funktionen müssen für jede eingebundene Funktion so implementiert werden, dass sie für den angefragten Webservice valide Nachrichten enthalten. Einer solchen Funktion werden die Parameter der Nachricht übergeben, als Rückgabewert entsteht eine Sequenz aus der generierten Nachricht und dem serverseitigen Funktionsnamen des Services. Diese Werte werden der Funktion `callService` übergeben, die die HTTP-Kommunikation mit dem Webservice übernimmt: Zunächst wird die Nachricht durch Aufruf der Funktion `soapRequest` (Zeile 29) in einen SOAP-Envelope verpackt und schließlich durch Aufruf der Funktion `soapPost` (Zeile 39) an den Webservice gesendet. Die Antwortnachricht des Webservices wird zurückgegeben. Der Aufruf bspw. der Funktion `prepopWS` des Webservices hat somit folgende Form:

```
app-ws:callService(app-ws:prepopWS("Nachrichteninhalt"))
```

```
1  xquery version "1.0";
2
3  module namespace app-ws = "app.ws";
4  import module namespace appUtil = "app.utils";
5
```

```
(:~ Template to generate the message for prepopulation data request
 : @param $token token to delete
 : @return sequence of message and function name
 :)
declare function app-ws:prepopWS($inner-content as xs:string) as item()+ {
      (<prep:getPrePopData>
        <prep:reqData>{$inner-content}
        </prep:reqData>
      </prep:getPrePopData>
    , 'getPrePopData')
};

(:~ Call web service
 : @param $content sequence auf message and function name, use output of
      template function
 : @return answer from web service
 :)
declare function app-ws:callService($content as item()*) as element()* {
  let $request := local:soapRequest($content[1])
  let $url := concat($appUtil:HISWrapperPath, $appUtil:HISWSLocalEndpoint,
        '/', $content[2])
  return local:soapPost($url, $request)
};

(: Construct a SOAP envelope around request-content and return it :)
declare function local:soapRequest($content as element()) as element() {
  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header/>
    <soap:Body>
      {$content}
    </soap:Body>
  </soap:Envelope>
};

(: Fire SOAP POST request to given url and return answer :)
declare function local:soapPost($url as xs:string, $request) as element()
    {
   httpclient:post(xs:anyURI($url),$request, false(),
   <headers><header name='Content-Type' value='application/soap+xml'/></
       headers>
   )
};
```

Programm A.19: Implementierung der Kommunikationsfunktion für SOAP-basierte Webservices.

A.3.5 Testausführung

Die hier gezeigte Implementierung gehört zu der in Abschnitt 5.1.5 beschriebenen Ausführung von Testfällen.

Programm A.20 zeigt das Modul zur Ausführung von Testfällen. Es basiert auf dem aus Abschnitt 5.1.1 bekannten Gerüst für View-Komponenten. Es enthält

von Zeile 44 bis Zeile 57 die Tabelle, in der die Testergebnisse dargestellt werden. In Zeile 9 wird die per Request-Parameter übergebene, gewünschte Gruppe von Tests aufgerufen, indem die Funktion `run-suite` des Models mit dem entsprechenden Argument aufgerufen wird. Das Resultat wird der Variablen `$unittest` zugeordnet und für jedes Testergebnis wird in den folgenden Zeilen ein Tabelleneintrag erzeugt. Entscheidend ist der in Zeile 25 ausgewertete Status des einzelnen Tests, der über Erfolg oder Misserfolg informiert. Alle Tabellenzeilen werden an die Variable `$content` gebunden und in Zeile 55 in das HTML-Gerüst eingesetzt.

```
1  xquery version "1.0";$
2
3  import module namespace app-html = "app.html" at "helper/html.xql";
4  import module namespace app-unittesting = "app.unittesting" at "../
       unittesting.xql";
5  import module namespace appUtil = "app" at "../../helper/helper.xql";
6
7  let $testsuite := request:get-parameter("suite","")
8  let $content :=
9    let $unittest := pruritus-unittesting:run-suite($testsuite)
10   for $test in $unittest//test
11   return
12   <tr>
13     <td>
14       <code>{$test/name/string()}</code>
15     </td>
16     <td><p><small>{$test/description/string()}</small></p></td>
17     <td>
18       <pre>{$test/expectation/string()}</pre>
19     </td>
20     <td>
21       <pre>{$test/result/string()}</pre>
22     </td>
23     <td>
24       {
25       if($test/status/string() eq "success") then
26         <span class="label label-success">Success</span>
27       else
28         <span class="label label-important">Fail</span>
29       }
30     </td>
31   </tr>
32
33 return
34 <html>
35   { app-html:htmlheader() }
36   <body>
37     <div class="container">
38       <div class="content">
39         <div class="page-header">
40           <h1>Unittesting <small>Suite: {$testsuite}</small></h1>
41         </div>
42         <div class="row">
43           <div class="span12">
```

```
44            <table class="table table-striped">
45                <thead>
46                  <tr>
47                    <th>Test for function:</th>
48                    <th>Description</th>
49                    <th>Expectation</th>
50                    <th>Result</th>
51                    <th>Status</th>
52                  </tr>
53                </thead>
54                <tbody>
55                { $content }
56                </tbody>
57              </table>
58            </div>
59          </div>
60        </div>
61        { app-html:htmlfooter() }
62      </div>
63    </body>
64  </html>
```

Programm A.20: View-Komponente zum Starten der Unittests.

Programm A.21 zeigt das zugehörige Bibliotheksmodul `app-unittesting` des Models, das auf dem in Abschnitt 5.1.1 gezeigten Gerüst basiert. Zunächst werden die Module der Testfälle importiert, in diesem Beispiel sind dies nur die Tests `testsuite-model-user` für die zu der Nutzerverwaltung gehörenden Module der Geschäftslogik. Weitere Testfälle sind dort ebenfalls zu importieren. Die Funktion `run-suite` in Zeile 16 führt den per Parameter übergebenen Testfall aus. Zunächst wird das System vorbereitet. Dazu wird mit der Funktion `backup-db` in Zeile 17 ein Backup des aktuellen Zustandes angelegt und folgend der Ausgangszustand für die Testdurchführung durch die Funktion `initialize-testscenario` hergestellt. Die Funktionen stammen aus dem Modul `app-install`, das hier nicht vorgestellt wird, da es sehr anwendungsspezifische Kopier- und Konfigurationsoperationen zum Installieren verschiedener Systemzustände enthält. Sodann wird in Zeile 22 der zum Parameter passende Testfall ausgewählt und ausgeführt. Das Ergebnis wird als Wert des Funktionsaufrufs zurückgegeben. Im Beispiel ist hier nur eine Testgruppe zu sehen, generell stehen alle importierten Module zur Verfügung. Schließlich wird der ursprüngliche Systemzustand wiederhergestellt, indem die Funktionen `recover-collection` und `logout` mit leeren Rückgabewerten in Zeile 26 aufgerufen werden.

```
1  xquery version "1.0";
2  module namespace  app-unittesting = "app.unittesting";
3
4  import module namespace app-html = "app.html";
5  import module namespace app-auth = "app.auth";
```

```
6  import module namespace appUtil = "app";
7  import module namespace app-install = "app.install";
8
9  import module namespace testsuite-model-user = "app.testsuite.model.user";
10 (: more import statements here for other testsuites :)
11
12 (:~ run a testsuite
13  : @param name of testsuite
14  : @return Results HTML-formatted
15  :)
16 declare function app-unittesting:run-suite($testsuite){
17 let $backup-collection := app-install:backup-db("testbackup")
18 return(
19   let $tmp :=app-install:initialize-testscenario()
20   let $tmp :=app-auth:login-user("Testuser", "testpassword")
21   return
22     if($testsuite eq "user-model") then
23       testsuite-model-user:unit-test-user()
24     (: if-then-else contructs for other testsuites :)
25     else ()
26   , app-install:recover-db($backup-collection), app-auth:logout()
27   )
28 };
```

Programm A.21: Model-Komponente zur Ausführung der Unittests.

Programm A.22 zeigt exemplarisch zwei Testfälle eines Moduls zur Benutzerverwaltung. Das Resultat der Funktionsausführung wird jeweils der Variablen `$result` zugewiesen, das erwartete Ergebnis der Variablen `$expectation`. Eine Hilfsfunktion aus der in Programm A.23 wiedergegebenen und noch zu erläuternden Bibliothek `app-testsuite-asserts` nimmt den Vergleich vor und erstellt das Ergebnis. Der erste Test in Zeile 13 prüft, ob die Funktion `get-all-users` zum Laden aller User-Elemente aus der Datenbank dieselben Elemente liefert wie das Lesen aus einer hinterlegten Datei mit Referenzdaten. Der zweite Test in Zeile 20 prüft, ob die Uhrzeit des letzten Logouts korrekt gesetzt und gelesen werden kann. Dazu wird zunächst die aktuelle Zeit bestimmt und mit der Funktion `set-last-logout` gesetzt. Dann wird versucht, diesen Wert mit der Funktion `get-last-logout` zu lesen. Der gelesene Wert wird mit dem zuvor bestimmten Wert verglichen.

```
1  xquery version "1.0";
2  module namespace app-testsuite-model-user = "app.testsuite.model.user";
3
4  import module namespace app-testsuite-asserts = "app.testsuite.asserts";
5  import module namespace app-model-user = "app.model.user";
6
7  (:~ execute the tests for user-model
8   : @return results
9   :)
10 declare function app-testsuite-model-user:unit-test-user()
11 {
12   <unittests>
```

```
13    {
14      let $result := app-model-user:get-all-users()
15      let $expectation := util:parse(file:read(concat($x4TUtil:localdir, '/
            testcases/data/backup/users.xml'))))/auth/users/user
16      return
17        app-testsuite-asserts:assert-equal("app-model-user:get-all-users()",
18          "Test if all users can be retrieved", $result, $expectation)
19    }
20    {
21      let $expectation := fn:current-dateTime()
22      let $set-logout := app-model-user:set-last-logout("Testuser", $
            expectation)
23      let $result := app-model-user:get-last-logout("Testuser")
24      return
25        app-testsuite-asserts:assert-equal-string("app-model-user:set-last-
              logout($user-name, $time-stamp), app-model-user:set-last-logout
              ($user-name)",
26          "Test set and get last logout", string($result), string($
                expectation))
27    }
28    </unittests>
29  };
```

Programm A.22: Testfälle eines Moduls zur Benutzerverwaltung.

Die bereits erwähnte Bibliothek `app-testsuite-asserts`, die sich in Programm A.23 befindet, stellt Vergleichsmethoden für unterschiedliche Datentypen bereit. Hier sind die Vergleiche für Elemente und Strings zu sehen, weitere sind analog ergänzbar. Die Funktion `scaffold` in Zeile 5 liefert das XML-Gerüst, das durch die View-Komponente in HTML transformiert wird. Dieses Gerüst wird von den für die Vergleiche zuständigen Funktionen `assert-equal` in Zeile 27 und `assert-equal-string` in Zeile 40 mit Werten versehen zurückgegeben. Abbildung A.1 zeigt das Ergebnis der Testdurchführung als Webseite.

```
1   xquery version "1.0";
2
3   module namespace pruritus-testsuite-asserts = "pruritus.testsuite.asserts"
        ;
4   (: build XML scaffold for test results :)
5   declare function local:scaffold($name, $description, $result, $expected, $
        testresult){
6     <test>
7       <name>{$name}</name>
8       <description>{$description}</description>
9       <expectation>{$expected}</expectation>
10      <result>{$result}</result>
11      <status>
12        {
13        if($testresult) then "success"
14        else "fail"
15        }
16      </status>
17    </test>
```

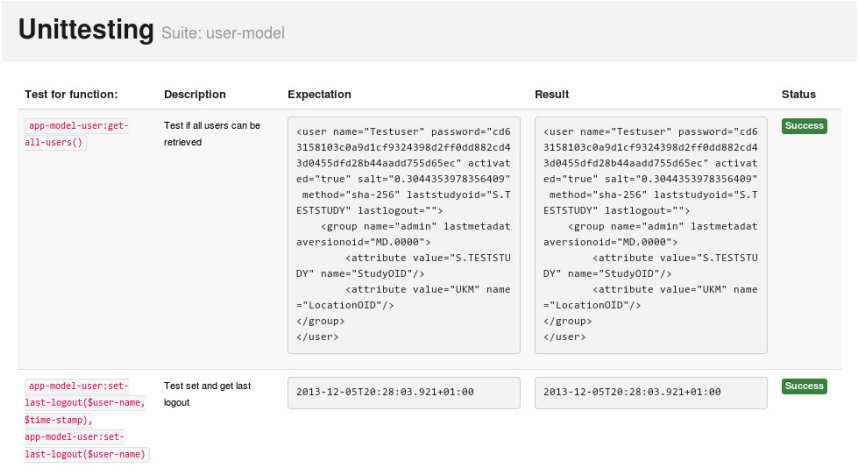
```
18  };
19
20  (:~ equal assert for elements
21   : @param $name name of the test
22   : @param $description description of the test
23   : @param $result actual result of the test execution
24   : @param $expected expected result of the test execution
25   : @return XML Fragment according to scaffold
26  :)
27  declare function pruritus-testsuite-asserts:assert-equal($name as
        xs:string, $description as xs:string?, $result as element()*, $
        expected as element()*)
28    $result, $expected)
29  {
30    local:scaffold($name, $description, util:serialize($result,()),
          util:serialize($expected,()), (util:serialize($result,()) eq
          util:serialize($expected,())))
31  };
32
33  (:~ equal assert for strings
34   : @param $name name of the test
35   : @param $description description of the test
36   : @param $result actual result of the test execution
37   : @param $expected expected result of the test execution
38   : @return XML Fragment according to scaffold
39  :)
40  declare function pruritus-testsuite-asserts:assert-equal-string($name as
        xs:string, $description as xs:string?, $result as xs:string?, $
        expected as xs:string?)
41
42    $result as xs:string?, $expected as xs:string?)
43  {
44    local:scaffold($name, $description, $result, $expected, ($result eq $
          expected))
45  };
```

Programm A.23: Hilfsbibliothek zum Ausführen der Testvergleiche und Erzeugen der Resultate.

A.4 Implementierungsbeispiele aus x4T

A.4.1 Anfordern der Vorbelegungsdaten

Die in Programm A.24 gezeigte Funktion `get-prepop-data` ist für die Anforderung der Vorbelegungsdaten zuständig. Dazu wird mit den ODM-Metadaten und den angegebenen Parametern das in Programm A.25 gezeigte XSLT-Programm aufgerufen. Die damit generierte Nachricht, für die ein Beispiel in Programm A.26 gegeben ist, wird als Anfrage an den KIS-Adapter geschickt. Die erhaltene Antwort bildet den Rückgabewert der Funktion.



Unittesting Suite: user-model

Test for function:	Description	Expectation	Result	Status
app-model-user:get-all-users()	Test if all users can be retrieved	<user name="Testuser" password="cd63158103c0a9d1cf9324398d2ff0dd882cd43d0455dfd28b44aadd755d65ec" activated="true" salt="0.3044353978356409" method="sha-256" laststudyoid="S.TESTSTUDY" lastlogout=""> <group name="admin" lastmetadataversionoid="MD.0000"> <attribute value="S.TESTSTUDY" name="StudyOID"/> <attribute value="UKM" name="LocationOID"/> </group> </user>	<user name="Testuser" password="cd63158103c0a9d1cf9324398d2ff0dd882cd43d0455dfd28b44aadd755d65ec" activated="true" salt="0.3044353978356409" method="sha-256" laststudyoid="S.TESTSTUDY" lastlogout=""> <group name="admin" lastmetadataversionoid="MD.0000"> <attribute value="S.TESTSTUDY" name="StudyOID"/> <attribute value="UKM" name="LocationOID"/> </group> </user>	Success
app-model-user:set-last-logout($user-name, $time-stamp), app-model-user:set-last-logout($user-name)	Test set and get last logout	2013-12-05T20:28:03.921+01:00	2013-12-05T20:28:03.921+01:00	Success

Abbildung A.1: Als Webseite ausgegebenes Ergebnis der hier gezeigten Testfälle.

```
1  (:~ Constructs prepopulation data request for a subject, triggers WS
       request, and writes received data into x4t namespace for ItemData
2  : @param $study-oid study oid for which the data is needed
3  : @param $pseudonymid subject whose prepopdata is requested
4  : @param $metadataversion-oid corresponding MetaDataVersionOID
5  :)
6  declare function x4T-patient:get-prepop-data($study-oid, $pseudonymid, $
       metadataversion-oid)
7  {
8    (: read study :)
9    let $study-data := x4T-model-study:get-study($study-oid)
10   (: creation of the request message content, parameters to configure the
         XSLT sheet:)
11   let $request-content := util:serialize(
12     transform:transform($study-data, xs:anyURI("patient/prepopRequest.xsl"
           ),
13     <parameters>
14       <param name="metadataversion-oid" value="{$metadataversion-oid}"/>
15       <param name="pseudonymid" value="{$pseudonymid}"/>
16       <param name="timestamp" value="{fn:current-dateTime()}"/>
17       <param name="name" value="x4t"/>
18     </parameters>),
19     "method=xml media-type=text/plain omit-xml-declaration=yes")
20   (: call webservice wrapper :)
21   let $httpanswer := x4T-ws:callService(x4T-ws:prepopWS($request-content))
22   (: parse answer as XML and return :)
23   return
24     util:parse($httpanswer)
```

```
25  };
```

Programm A.24: Anfordern der Vorbelegungsdaten vom KIS-Adapter.

```
1   <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2     version="2.0" xmlns="http://www.cdisc.org/ns/odm/v1.3"
3     xpath-default-namespace="http://www.cdisc.org/ns/odm/v1.3"
4     xmlns:fn="http://www.w3.org/2005/xpath-functions"
5     xmlns:odm="http://www.cdisc.org/ns/odm/v1.3">
6     <xsl:output method="xml" indent="yes" />
7     <!-- generates the prepopulation requests all ItemDefs in a study -->
8     <xsl:param name="metadataversion-oid"/>
9     <xsl:param name="pseudonymid"/>
10    <xsl:param name="timestamp"/>
11    <xsl:param name="name"/>
12
13    <xsl:template match="ODM"><xsl:apply-templates/></xsl:template>
14    <!-- creates the header -->
15    <xsl:template match="Study" priority="9">
16      <xsl:variable name="StudyOID" select="@OID"/>
17      <xsl:element name="ClinicalDataRequest" namespace="">
18        <xsl:attribute name="rid"><xsl:value-of select="$rid"/></
              xsl:attribute>
19        <xsl:attribute name="timestamp"><xsl:value-of select="$timestamp"/><
              /xsl:attribute>
20        <xsl:element name="Enquirer" namespace="">
21          <xsl:element name="Name" namespace=""><xsl:value-of select="$name"
                /></xsl:element>
22        </xsl:element>
23        <xsl:element name="PatientInfo" namespace="">
24          <xsl:element name="Pseudonymid" namespace="">
25            <xsl:value-of select="$pseudonymid"/>
26          </xsl:element>
27        </xsl:element>
28        <!-- single request items go here -->
29        <xsl:element name="RequestSet" namespace="">
30          <xsl:for-each select="MetaDataVersion[@OID = $metadataversion-oid
                ]/ItemDef[exists(Alias)]">
31          <xsl:variable name="count"><xsl:number level="single"></xsl:number
                ></xsl:variable>
32          <xsl:call-template name="transformAliases">
33            <xsl:with-param name="count" select="$count">
34            </xsl:with-param>
35          </xsl:call-template>
36        </xsl:for-each>
37      </xsl:element>
38      </xsl:element>
39    </xsl:template>
40    <!-- extract annotations and construct single request items -->
41    <xsl:template name="transformAliases">
42      <xsl:param name="count"/>
43      <xsl:element name="ReqConceptCodes" namespace="">
44        <xsl:attribute name="oid">
45          <xsl:value-of select="$count"></xsl:value-of>
46        </xsl:attribute>
47        <xsl:for-each select="Alias">
48          <xsl:element name="ReqCodeItem" namespace="">
```

```
49            <xsl:attribute name="code" select="@Name"/>
50            <xsl:attribute name="codesystemoid" select="@Context"/>
51          </xsl:element>
52        </xsl:for-each>
53      </xsl:element>
54    </xsl:template>
55  </xsl:stylesheet>
```

Programm A.25: XSLT-Programm zur Erzeugung von Nachrichten an den KIS-Adapter zur Abfrage von Vorbelegungsdaten.

```
1  <ClinicalDataRequest timestamp="2013-11-28T15:52:19.369+01:00">
2      <Enquirer>
3          <Name>x4t</Name>
4      </Enquirer>
5      <PatientInfo>
6          <Pseudonymid>doe</Pseudonymid>
7      </PatientInfo>
8      <RequestSet>
9          <ReqConceptCodes oid="1">
10             <ReqCodeItem code="184099003" codesystemoid="SNOMED"/>
11         </ReqConceptCodes>
12         <ReqConceptCodes oid="2">
13             <ReqCodeItem code="C25347" codesystemoid="NCIt"/>
14         </ReqConceptCodes>
15         <ReqConceptCodes oid="3">
16             <ReqCodeItem code="C15220" codesystemoid="NCIt"/>
17             <ReqCodeItem code="C25372" codesystemoid="NCIt"/>
18         </ReqConceptCodes>…
19
20     </RequestSet>
21 </ClinicalDataRequest>
```

Programm A.26: Generierte Nachricht an den KIS-Adapter zur Abfrage von Vorbelegungsdaten.

A.4.2 Integrieren der Vorbelegungsdaten

Programm A.27 zeigt die Funktion zur Integration der Vorbelegungsdaten im dafür vorgesehenen und mit dem Präfix `prepop` bezeichneten Namensraum. Die Vorbelegungsdaten werden mit der aus Abschnitt A.4.1 bekannten Funktion `get-prepop-data` angefordert. Dann werden die erhaltenen Daten anhand der semantischen Annotationen den zugehörigen `ItemData`-Elementen des ODM zugeordnet und, falls sie neu sind, dort eingefügt.

```
1  (:~ Asks for prepopulation data for a subject and writes the data into x4t
       prepop namespace for ItemData
2   : @param $subject-key SubjectKey of corresponding user
3   : @param $study-oid corresponding StudyOID
4   : @param $metadataversion-oid corresponding MetaDataVersionOID
5   :)
```

```
6  declare function local:prepopulate-form($subject-key, $study-oid, $
       metadataversion-oid)
7  {
8    (: subject to modify :)
9    let $subject-data := x4T-model-patient:get-patient($study-oid, $
         metadataversion-oid, $subject-key)
10   return
11   (: set write lock for the rest of the function :)
12     util:shared-lock($subject-data,
13     (: return error, if data is not present :)
14     (if (not($subject-data)) then
15       <error>study or subject not found</error>
16     else
17       (: get prepopulation data :)
18       let $prepop-data := x4T-patient:get-prepop-data($study-oid, $subject
             -key, $metadataversion-oid)
19       (: iterate the received items :)
20       for $response-item in $prepop-data/ClinicalValueResponse/ResponseSet
             /ReqConceptCodes
21       (: iterate the responed values in every item :)
22       for $response-value in $response-item/ResponseValues/ResVal
23       (: find the semantically correct insertion positions :)
24       for $insert-place in $subject-data/odm:StudyEventData/odm:FormData/
             odm:ItemGroupData/prepop:prepopDataSet/prepop:codeData
25       (: every code in ODM must be present in response-item in order to
             fit the semantic and write prepopvalue to prepopnamespace in
             ODM:)
26       where (every $codesystem in $insert-place/prepop:codesystemoid
             satisfies $codesystem = $response-item/ResCodeItem/
             @codesystemoid) and (every $code in $insert-place/prepop:code
             satisfies $code = $response-item/ResCodeItem/@code)
27       return
28       (: convert date and time separated by Clinical Interface to dateTime
              datatype :)
29         let $entryDate := try{fn:dateTime($response-value/
               @investigationDate,$response-value/@investigationTime)} catch
                * {()}
30         (: construct the XML fragment of prepop namespace to store the
              prepopulation data :)
31         let $insert-content :=
32           <prepop:prepopDataItem>
33             <prepop:formName>{$response-value/@formName/string()}</
                   prepop:formName>
34             <prepop:sourceInstanceID>{$response-value/@formInstanceID/
                   string()}</prepop:sourceInstanceID>
35             <prepop:entryDate>{$entryDate}</prepop:entryDate>
36             <prepop:value>{$response-value/@value/string()}</prepop:value>
37           </prepop:prepopDataItem>
38         return
39           (: do not change the document, if entry date is invalid. Log an
                error instead :)
40           if (empty($entryDate)) then
41             util:log('debug', $response-value/@investigationDate || $
                   response-value/@investigationTime || "ERROR, because no
                   DateTime creatable from")
42           else
```

```
43              (: check if identical prepop data set is already present, then
                     do nothing:)
44              if (exists($insert-place/prepop:prepopDataItem[
                     prepop:entryDate=fn:dateTime($response-value/
                     @investigationDate, $response-value/@investigationTime)])
                      and $insert-place/prepop:prepopDataItem/prepop:value=$
                     response-value/@value) then
45                ()
46              (: insert constructed prepop data:)
47              else
48                update insert $insert-content into $insert-place
49      (:return success:)
50      ,<success>Prepopulation done</success>
51      )
52    )
53  };
```

Programm A.27: Abfragen der Vorbelegungsdaten aus dem KIS und Schreiben der Werte in die Schemaerweiterung x4T.

A.4.3 Automatisch Dokumentieren

Die Funktion für automatisches Dokumentieren kann im KIS ausgelöst werden, wenn der dokumentierende Arzt die für ein Studienformular relevanten und bereits im KIS dokumentierten Werte ohne weitere Überprüfung in die Studiendokumentation übertragen möchte. In diesem Fall führt die in Programm A.28 gezeigte Funktion die Vorbelegungsfunktion aus und ruft im Anschluss die Funktion `copy-prepop-to-value` auf, die das Kopieren der Werte in das `Value`-Attribut des Elements `ItemData` übernimmt. Neben dem Kopieren wird eine Signatur eingefügt, die Datum, Herkunft und den die automatische Dokumentation veranlassenden Benutzer dokumentiert. Für Formulare, die wiederholt ausgefüllt werden können, wird neben den semantischen Metadaten des `DataItems` auch noch die Identität des zugehörigen Quelldokuments überprüft.

```
1   (:~
2   : Documents a patient automatically, i.e. gets prepopdata and writes
        prepopdata into ItemData
3   : @param $task the autodocumentation information as received from HIS:
4   : <TokenResponse function="AutoPatientDocumentation" token="34797600">
5   :     <TParam pname="study_oid" pvalue="S.PRURITUS"/>
6   :     <TParam pname="metadataversion_oid" pvalue="MD.0000"/>
7   :     <TParam pname="psd" pvalue="doe"/>
8   :     <TParam pname="user_oid" pvalue="mustermann"/>
9   : </TokenResponse>
10  :)
11  declare function x4T-patient:auto-patient-documentation($task){
12  (:
13  These items are given by HIS. At first, the FormInfoWebservice is asked
        about forms that are present for a subject. Then, it is checked which
```

```
        forms are missing in formhandler and those are created. All forms
       are prepopulated then.
14  Prepop values are copied afterwards. At the end, the clincal interface is
       informed by a confirmation service
15  :)
16  let $token := $task/@token/string()
17  let $study-oid := $task/TParam[@pname/string() = "study_oid"]/@pvalue/
       string()
18  let $metadataversion-oid := $task/TParam[@pname/string() = "
       metadataversion_oid"]/@pvalue/string()
19  let $metadataversion := x4T-model-metadataversion:get-metadataversion($
       study-oid, $metadataversion-oid)
20  let $subject-key := $task/TParam[@pname/string() = "psd"]/@pvalue/string()
21  let $user := $task/TParam[@pname/string() = "user_oid"]/@pvalue/string()
22
23  let $subject-data := x4T-model-patient:get-patient($study-oid, $
       metadataversion-oid, $subject-key)
24
25  (: Create missing form repetitions :)
26  let $tmp := local:create-missing-forms($subject-key, $study-oid, $
       metadataversion-oid)
27
28  (: Do common prepopulation in order to fill the newly added form
       repetitions :)
29  let $tmp := local:prepopulate-form($subject-key, $study-oid, $
       metadataversion-oid)
30
31  (: call the copy function and acknowlegde the Clinical Interface:)
32  return (local:copy-prepop-to-value($subject-data, $user),
33    x4T-ws:callService(x4T-ws:confirmAutoImportWS($token, fn:true())))
34  };
35
36
37  (:~ Copy Prepop values from the prepop-value section and namespace into
       value
38   : section of a subject. Needed for automatic documentation.
39   : For non-repeating forms, the latest value is copied. For repeating
         forms, values from the respective sourceInstanceID are copied.
40   : @param $subject-data SubjectData element from ODM
41   : @param $user user oid of triggering user
42   :)
43  declare function local:copy-prepop-to-value($subject-data as element(), $
       user as xs:string){
44  (: For forms that are present :)
45  for $form in $subject-data/odm:StudyEventData/odm:FormData
46  let $metadataversion := $subject-data/..
47  let $studyevent-oid := $form/../@StudyEventOID/string()
48  let $studyevent-repeatkey := $form/../@StudyEventRepeatKey/string()
49  let $form-oid := $form/@FormOID/string()
50  let $form-repeatkey := $form/@FormRepeatKey/string()
51  (: only repeating forms have sourceInstanceData :)
52  let $sourceInstanceID := $form/prepop:sourceInstanceData
53
54  (: iterate over itemgroups [because ItemOID is unique within itemgroup].
       If parameters are given, they are respected :)
55  for $itemGroupData in $subject-data/odm:StudyEventData
```

```
56    [if (empty($studyevent-oid)) then true() else @StudyEventOID = $
          studyevent-oid]
57      [if (empty($studyevent-repeatkey)) then true() else
            @StudyEventRepeatKey = $studyevent-repeatkey]/odm:FormData
58      [if (empty($form-oid)) then true() else @FormOID = $form-oid]
59      [if (empty($form-repeatkey)) then true() else @FormRepeatKey = $form-
            repeatkey]
60      /odm:ItemGroupData
61      (: iterate  over prepopvalues present in the ItemGroup :)
62      for $prepopData in (if ($sourceInstanceID) then
63        $itemGroupData/prepop:prepopDataSet[prepop:codeData/
              prepop:prepopDataItem[prepop:sourceInstanceID eq $
              sourceInstanceID]/prepop:value]
64        else $itemGroupData/prepop:prepopDataSet[prepop:codeData/
              prepop:prepopDataItem]
65        )
66        return
67          let $item-oid := $prepopData/@ItemOID
68          return
69            (
70            (: if sourceInstance is given, the specific value is taken,
                  otherwise the newest :)
71            let $prepop-raw-value :=
72              if ($sourceInstanceID)
73              (:it is necessary to check for the latest date entry in
                    addition to sourceInstanceID:)
74              then  $prepopData[@ItemOID = $item-oid]/prepop:codeData/
                    prepop:prepopDataItem[prepop:sourceInstanceID eq $
                    sourceInstanceID][xs:dateTime(prepop:entryDate) eq fn:max
                    (../prepop:prepopDataItem[prepop:sourceInstanceID eq $
                    sourceInstanceID]/prepop:entryDate/xs:dateTime(.))]/
                    prepop:value
75              else  $prepopData[@ItemOID = $item-oid]/prepop:codeData/
                    prepop:prepopDataItem[xs:dateTime(prepop:entryDate) eq
                    fn:max(../prepop:prepopDataItem/prepop:entryDate/
                    xs:dateTime(.))]/prepop:value
76            (: check if value is a boolean and needs transformation :)
77            let $prepop-value :=
78              if ($metadataversion/odm:ItemDef[@OID=$item-oid]/@DataType = "
                    boolean")
79              then xs:boolean($prepop-raw-value[1]/string()) (: there are as
                     many prepopdata entries as repetitions. As they are
                    equal, take the first :)
80              else $prepop-raw-value[1]/string()
81            return (: write prepop value into ItemData :)
82            (update value $itemGroupData/odm:ItemData[@ItemOID = $item-oid]/
                  @Value with $prepop-value,
83            (: delete any existing Signatures to make room for the new one:)
84            update delete $itemGroupData/odm:ItemData[@ItemOID = $item-oid]/
                  odm:Signature,
85            update insert  <odm:Signature><odm:UserRef UserOID="{$user}" /><
                  odm:LocationRef LocationOID="x4T"/><odm:SignatureRef
                  SignatureOID="AutoDocumentation"/><odm:DateTimeStamp>{
                  xs:dateTime($prepop-raw-value[1]/../prepop:entryDate/string
                  ())}</odm:DateTimeStamp></odm:Signature> into $
                  itemGroupData/odm:ItemData[@ItemOID = $item-oid]),
```

```
86              if ($itemGroupData/odm:ItemData[@ItemOID = $item-oid]/
                     prepop:Source)
87              then
88              (: Data provenance. Again, if there is sourceInstanceID in this
                     iteration, then sourceInstanceID ist identified by that,
                     otherwise newest is used; consistent with the selection of
                     the value above :)
89                  if ($sourceInstanceID)
90                  then
91                    update value $itemGroupData/odm:ItemData[@ItemOID = $item-
                           oid]/prepop:Source/@InstanceID with $prepopData/
                           prepop:codeData/prepop:prepopDataItem[
                           prepop:sourceInstanceID eq $sourceInstanceID]/
                           prepop:sourceInstanceID
92                  else
93                    update value $itemGroupData/odm:ItemData[@ItemOID = $item-
                           oid]/prepop:Source/@InstanceID with $prepopData/
                           prepop:codeData/prepop:prepopDataItem[xs:dateTime(
                           prepop:entryDate) eq fn:max(../prepop:prepopDataItem/
                           prepop:entryDate/xs:dateTime(.))]/
                           prepop:sourceInstanceID
94              else
95                if
96                  ($sourceInstanceID)
97                then
98                  update insert <prepop:Source InstanceID="{$prepopData/
                         prepop:codeData/prepop:prepopDataItem[
                         prepop:sourceInstanceID eq $sourceInstanceID]/
                         prepop:sourceInstanceID}"/> into $itemGroupData/
                         odm:ItemData[@ItemOID = $item-oid]
99                else
100                 update insert <prepop:Source InstanceID="{$prepopData/
                        prepop:codeData/prepop:prepopDataItem[xs:dateTime(
                        prepop:entryDate) eq fn:max(../prepop:prepopDataItem/
                        prepop:entryDate/xs:dateTime(.))]/
                        prepop:sourceInstanceID}"/> into $itemGroupData/
                        odm:ItemData[@ItemOID = $item-oid]
101             )
102 };
```

Programm A.28: Funktion, die die automatische Dokumentation ausführt.

A.4.4 Formulardarstellung

Das Hauptmodul in Programm A.29 löst als View-Komponente die dynamische Erzeugung des Dokumentationsformulars und die Einbettung in die generierte Webseite aus. Das XForms-Formular wird durch ein parametrisiertes XSLT-Programm erzeugt, der HTML-Code wird größtenteils über Template-Funktionen eingebunden. Programm A.30 zeigt die Kernfunktionalität des XSLT-Programms. Abbildung A.2 auf Seite 265 zeigt ein damit generiertes Formular.

```
1  xquery version "1.0";
2
```

```
3  (: extract parameters from request :)
4  let $patient-oid := request:get-parameter("patient_oid", "")
5  let $study-oid := request:get-parameter("study_oid", "")
6  let $metadataversion-oid := request:get-parameter("metadataversion_oid","")
7  let $studyevent-oid := request:get-parameter("studyevent_oid", "")
8  let $studyevent-repeatkey:= request:get-parameter("studyevent_repeatkey",
       "")
9  let $form-oid := request:get-parameter("form_oid","")
10 let $form-repeatkey := request:get-parameter("form_repeatkey", "")
11 (: generation of XForms content :)
12 let $content :=
13   let $study := x4T-model-study:get-study($study-oid)
14   return
15   transform:transform($study, xs:anyURI("../odm/xformsFromStudy.xsl"), (
16     <parameters>
17       <param name="patient-oid" value="{$patient-oid}"/>
18       <param name="study-oid" value="{$study-oid}"/>
19       <param name="metadataversion-oid" value="{$metadataversion-oid}"/>
20       <param name="studyevent-oid" value="{$studyevent-oid}"/>
21       <param name="studyevent-repeatkey" value="{$studyevent-repeatkey}"/>
22       <param name="form-oid" value="{$form-oid}"/>
23       <param name="form-repeatkey" value="{$form-repeatkey}"/>
24     </parameters>))
25
26 return
27 <html>
28   {
29     x4T-html:htmlheader()
30   }
31   <body>
32     <div class="container">
33       <div class="content">
34         <div class="page-header">
35           {x4T-html:headerright()}
36           {x4T-html:format-headline("Formular", "editieren")}
37         </div>
38         {
39           x4T-html:headerline("Daten", "", "Patientenliste", concat($
                 x4TUtil:rootURL, "/data/patients?study_oid=", $study-oid, "
                 &amp;metadataversion_oid=", $metadataversion-oid), concat("
                 Patienten-ID: ", $patient-oid) )
40         }
41         {
42           $content
43         }
44       </div>
45       {
46         x4T-html:htmlfooter()
47       }
48     </div>
49   </body>
50 </html>
```

Programm A.29: Implementierung der View-Komponente, die ein Dokumentationsformular anzeigt.

```
1  <xsl:stylesheet>
2  <!-- this programm transforms a form of an ODM Study for one patient into
       XForms -->
3
4    <!-- received parameters and variables containing text for
         internationalization removed to keep example shorter -->
5
6    <!-- XForms model is generated at highest level -->
7    <xsl:template match="Study">
8      <xsl:variable name="STUDYOID" select="$study-oid"/>
9      <xf:model id="m-study">
10       <xf:instance id="i-data" xmlns="http://www.cdisc.org/ns/odm/v1.3"
             xmlns:odm="http://www.cdisc.org/ns/odm/v1.3"><data/>
11       </xf:instance>
12       <xf:submission id="s-get-i-data" resource="{$localpath}/patient/
             getClinicalData.xql?{$URLparameters}" method="get" replace="
             instance" instance="i-data">
13       </xf:submission>
14       <xf:instance id="i-study" xmlns="http://www.cdisc.org/ns/odm/v1.3"
             xmlns:odm="http://www.cdisc.org/ns/odm/v1.3"><data/>
15       </xf:instance>
16       <xf:submission id="s-get-i-study" resource="{$localpath}/odm/show.
             xql?study_oid={$study-oid}" method="get" replace="instance"
             instance="i-study" mediatype="application/xml">
17       </xf:submission>
18       <!-- load data after form is ready -->
19       <xf:action ev:event="xforms-ready">
20         <xf:send submission="s-get-i-study"/>
21         <xf:send submission="s-get-i-data"/>
22       </xf:action>
23       <!-- to store form data -->
24       <xf:submission id="s-save" method="post" resource="{$saveURL}"
             replace="none" instance="i-data"/>
25
26       <!-- bind types-->
27       <xf:bind nodeset="//odm:FormData/odm:ItemGroupData/odm:ItemData[
             @ItemOID = instance('i-study')/odm:MetaDataVersion/odm:ItemDef[
             @DataType='partialDate' or @DataType='date']/@OID]/@Value" type
             ="xf:date"/>
28       <xf:bind nodeset="//odm:ItemData[@ItemOID = instance('i-study')/
             odm:MetaDataVersion/odm:ItemDef[@DataType='integer']/@OID]/
             @Value" type="xf:integer"/>
29       <xf:bind nodeset="//odm:ItemData[@ItemOID = instance('i-study')/
             odm:MetaDataVersion/odm:ItemDef[@DataType='boolean']/@OID]/
             @Value" type="xf:boolean"/>
30       <!-- some binds removed to keep example short -->
31     </xf:model>
32     <!-- program traverses the hierarchy of MetaDataVersion/Protocol/
           StudyEvent/Form/ItemGroup/Item-->
33     <xsl:apply-templates select="MetaDataVersion[@OID=$metadataversion-oid
           ]"/>
34   </xsl:template>
35
36   <xsl:template match="MetaDataVersion">
37     <div>
38       <hr/>
39       <xsl:apply-templates select="Protocol"/>
```

```
40      </div>
41    </xsl:template>
42
43    <xsl:template match="Protocol">
44      <div>
45        <!-- selection of StudyEvent -->
46        <xsl:for-each select="StudyEventRef[@StudyEventOID=$studyevent-oid]"
              >
47          <xsl:apply-templates select="//StudyEventDef[@OID=$studyevent-oid]
                "/>
48        </xsl:for-each>
49        <!-- The buttons at the bottom, some removed to keep example shorter
              -->
50        <xf:submit submission="s-save">
51          <xf:label><xsl:value-of select="$txtSubmit"/></xf:label>
52        </xf:submit>
53        <xf:trigger>
54          <xf:label><xsl:value-of select="$txtClose"/></xf:label>
55          <xf:load ev:event="DOMActivate" resource="javascript:history.back
                ()"/>
56        </xf:trigger>
57      </div>
58    </xsl:template>
59
60    <xsl:template match="StudyEventDef">
61      <xsl:variable name="OID" select="@OID"/>
62      <!-- selection of Form -->
63      <xsl:for-each select="//StudyEventDef[@OID=$OID]/FormRef[@FormOID=$
            form-oid]">
64        <xsl:apply-templates select="//FormDef[@OID=$form-oid]"/>
65      </xsl:for-each>
66    </xsl:template>
67
68    <xsl:template match="FormDef">
69      <xsl:param name="storagePath"/>
70      <xsl:variable name="OID" select="@OID"/>
71      <!-- selection of ItemGroup -->
72      <xsl:for-each select="./ItemGroupRef">
73        <xsl:variable name="IOID" select="@ItemGroupOID"/>
74        <xsl:apply-templates select="//ItemGroupDef[@OID=$IOID]">
75          <!-- constructs the XPath expression to reach an Value attribute.
                quotation mark encoding -->
76          <xsl:with-param name="storagePath" select="concat($storagePath, '/
                odm:FormData[@FormOID=',$quot,$OID,$quot,']')"/>
77        </xsl:apply-templates>
78      </xsl:for-each>
79    </xsl:template>
80
81    <xsl:template match="ItemGroupDef">
82      <xsl:param name="storagePath"/>
83      <xsl:variable name="OID" select="@OID"/>
84      <table>
85        <tbody>
86          <!-- selection of Item -->
87          <xsl:for-each select="ItemRef">
88            <xsl:sort select="@OrderNumber"/>
89            <xsl:variable name="IOID" select="@ItemOID"/>
```

```
90            <xsl:apply-templates select="//ItemDef[@OID=$IOID]">
91              <xsl:with-param name="storagePath" select="concat($storagePath
                    , '/odm:ItemGroupData[@ItemGroupOID=',$quot,$OID,$quot
                    ,']')"/>
92            </xsl:apply-templates>
93          </xsl:for-each>
94        </tbody>
95      </table>
96    </xsl:template>
97
98    <!-- renders a single line for data input-->
99    <xsl:template match="ItemDef">
100     <xsl:param name="storagePath"/>
101     <tr>
102       <td class="question">
103         <xf:group ref="{$storagePath}">
104           <xsl:call-template name="renderQuestion">
105             <xsl:with-param name="ItemOID" select="@OID"/>
106           </xsl:call-template>
107         </xf:group>
108       </td>
109       <td>
110         <xsl:call-template name="itemInput">
111           <xsl:with-param name="storagePath" select="$storagePath"/>
112         </xsl:call-template>
113       </td>
114     </tr>
115   </xsl:template>
116
117   <!-- renders the question-->
118   <xsl:template name="renderQuestion">
119     <xsl:param name="ItemOID"/>
120     <xsl:value-of select="//odm:ItemDef[@OID = $ItemOID]/odm:Question/
          odm:TranslatedText[1]"/>
121   </xsl:template>
122
123   <!-- renders the input element-->
124   <xsl:template name="itemInput">
125     <xsl:param name="storagePath"/>
126     <xsl:variable name="OID" select="@OID"/>
127     <xsl:variable name="destination"><xsl:value-of select="$storagePath"/>
          /odm:ItemData[@ItemOID='<xsl:value-of select="@OID"/>']/@Value</
          xsl:variable>
128
129     <!-- this renders the boolean input -->
130     <xsl:if test="@DataType='boolean'">
131       <xf:select1 ref="{$destination}" appearance="full" class="x4tInline"
              id="{$inputId_noprepop}">
132         <xf:item>
133           <xf:label><xsl:value-of select="$txtYes"/></xf:label>
134           <xf:value value="boolean-from-string('true')"></xf:value>
135         </xf:item>
136         <xf:item>
137           <xf:label><xsl:value-of select="$txtNo"/></xf:label>
138           <xf:value value="boolean-from-string('false()')"/>
139         </xf:item>
140       </xf:select1>
```

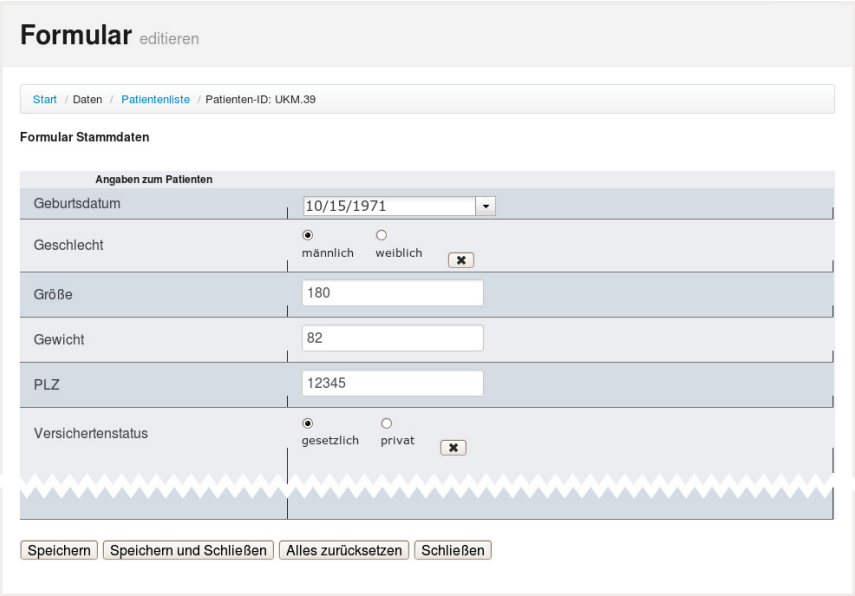


Abbildung A.2: Generiertes Dokumentationsformular für Stammdaten. Vertrauliche Bereiche sind ausgeblendet.

```
141       </xsl:if>
142
143       <!-- this renders the common single line input -->
144       <xsl:if test="@DataType='text' or @DataType='string'">
145         <xf:input ref="{$destination}" id="{$inputId_noprepop}"/>
146       </xsl:if>
147
148       <!-- other input renderings removed to keep example shorter -->
149     </xsl:template>
150
151       <!-- some auxiliary templates removed to keep example shorter -->
152   </xsl:stylesheet>
```

Programm A.30: XSLT-Programm, das aus einer ODM-Datei und übergebenen Parametern ein XForms-Formular generiert. Aus Gründen der Darstellbarkeit ist die hier gezeigte Version stark vereinfacht.

Quellenverzeichnisse

Literaturquellen

[ABS99] Serge Abiteboul, Peter Buneman und Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1999.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi A. Kuno und Vijay Machiraju. *Web Services - Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Berlin: Springer, 2004.

[AH02] Sergio Antoy und Michael Hanus. „Functional Logic Design Patterns". In: *Functional and Logic Programming*. Hrsg. von Zhenjiang Hu und Mario Rodríguez-Artalejo. Bd. 2441. Lecture Notes in Computer Science. Berlin: Springer, 2002, S. 67–87. DOI: **10.1007/3-540-45788-7_4**.

[AHKP09] Christian Arndt, Christian Hermanns, Herbert Kuchen und Michael Poldner. *Best Practices in der Softwareentwicklung*. Working Paper 1. Münster: Förderkreis der Angewandten Informatik an der Westfälischen Wilhelms-Universität Münster e. V., Feb. 2009. URL: **http://www.wi1.uni-muenster.de/pi/iai/publikationen/iai1.pdf**.

[AHM06] John Anvik, Lyndon Hiew und Gail C. Murphy. „Who should fix this bug?" In: *Proceedings of the 28th international conference on Software engineering*. ICSE '06. Shanghai, China: ACM, 2006, S. 361–370. DOI: **10.1145/1134285.1134336**.

[AL04] Marcelo Arenas und Leonid Libkin. „A normal form for XML documents". In: *ACM Trans. Database Syst.* 29.1 (März 2004), S. 195–232. DOI: **10.1145/974750.974757**.

[AL05] Marcelo Arenas und Leonid Libkin. „An information-theoretic approach to normal forms for relational and XML data". In: *J. ACM* 52.2 (März 2005), S. 246–283. DOI: **10.1145/1059513.1059519**.

[Arm06] Deborah J. Armstrong. „The quarks of object-oriented development". In: *Commun. ACM* 49.2 (Feb. 2006), S. 123–128. DOI: **10.1145/1113034.1113040**.

[Arr69] Kenneth J. Arrow. „The Organization of Economic Activity: Issues Pertinent to the Choice of Market versus Non-market Allocations". In: *Analysis and Evaluation of Public Expenditures: The PPB System* 1 (1969), S. 59–73.

[AS09] E. Ammenwerth und H-P. Spötl. „The Time Needed for Clinical Documentation versus Direct PatientCare". In: *Methods of Information in Medicine* 48.1 (2009), S. 84–91. DOI: **10.3414/ME0569**.

[Aut13] Autor unbekannt. „Open data: A new goldmine". In: *The Economist* 407.8836 (Mai 2013), S. 73. URL: **http://www.economist.com/news/business/21578084-making-official-data-public-could-spur-lots-innovation-new-goldmine**.

[Bab13] Deven Babre. „Clinical data interchange standards consortium: A bridge to overcome data standardisation". In: *Perspectives in Clinical Research* 4.2 (2013), S. 115–116. DOI: **10.4103/2229-3485.111779**.

[Bal01] Helmut Balzert. *Lehrbuch der Software-Technik: Software-Entwicklung*. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, 2001.

[Bal09] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Heidelberg: Spektrum Akademischer Verlag, 2009.

[Bal11] Helmut Balzert. *Lehrbuch der Software-Technik: Entwurf, Implementierung, Installation und Betrieb*. Heidelberg: Spektrum Akademischer Verlag, 2011.

[Ban88] François Banciihon. „Object-oriented database systems". In: *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. PODS '88. Austin, Texas, USA: ACM, 1988, S. 152–162. DOI: **10.1145/308386.308429**.

[Bas14] BaseX Team. *BaseX Documentation: Version 7.8*. Feb. 2014. URL: **http://files.basex.org/releases/7.8/BaseX78.pdf**.

[BB10] Jan Bosch und Petra Bosch-Sijtsema. „From integration to composition: On the impact of software product lines, global development and ecosystems". In: *Journal of Systems and Software* 83.1 (2010), S. 67–76. DOI: **10.1016/j.jss.2009.06.051**.

[BBC+10] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie und Jérôme Siméon. *XML Path Language (XPath) 2.0*. W3C Recommendation. W3C, Dez. 2010. URL: **http://www.w3.org/TR/2010/REC-xpath20-20101214/**.

[BCF+10] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie und Jérôme Siméon. *XQuery 1.0: An XML Query Language (Second Edition)*. W3C Recommendation. W3C, Dez. 2010. URL: **http://www.w3.org/TR/2010/REC-xquery-20101214/**.

[BCG+05] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad und Farouk Toumani. „Developing Adapters for Web Services Integration". In: *Advanced Information Systems Engineering*. Hrsg. von Oscar Pastor und João Falcão e Cunha. Bd. 3520. Lecture Notes in Computer Science. Berlin: Springer, 2005, S. 415–429. DOI: **10.1007/11431855_29**.

[BÇHL11] Bert Bos, Tantek Çelik, Ian Hickson und Håkon Wium Lie. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. W3C Recommendation. W3C, Juni 2011. URL: **http://www.w3.org/TR/2011/REC-CSS2-20110607**.

[BD13] Douglas K. Barry und David Dick. *Web Services, Service-Oriented Architectures, and Cloud Computing: The Savvy Manager's Guide*. Waltham, MA, USA: Morgan Kaufmann, 2013.

[BDE+02] Douglas Bryan, Vadim Draluk, Dave Ehnebuske u. a. *UDDI Version 2.04 API Specification*. UDDI Committee Specification. OASIS, Juli 2002. URL: **http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm**.

[BEL+12] Peter Brown, Jeff A. Estefan, Ken Laskey, Francis G. McCabe und Danny Thornton. *Reference Architecture Foundation for Service Oriented Architecture Version 1.0*. Committee Specification 01. OASIS, Dez. 2012. URL: **http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.pdf**.

[Ben12] Tim Benson. *Principles of Health Interoperability HL7 and SNOMED*. Second Edition. Health informatics. London: Springer-Verlag, 2012. URL: **http://www.springer.com/public+health/book/978-1-4471-2800-7**.

[Ber06] Anders Berglund. *Extensible Stylesheet Language (XSL) Version 1.1*. W3C Recommendation. W3C, Dez. 2006. URL: **http://www.w3.org/TR/2006/REC-xsl11-20061205/**.

[BETL12] Katrin Braunschweig, Julian Eberius, Maik Thiele und Wolfgang Lehner. „The State of Open Data – Limits of Current Open Data Platforms". In: *Proceedings of the 21st World Wide Web Conference 2012, Web Science Track at WWW'12, Lyon, France, April 16-20, 2012*. ACM, 2012.

[BFB+14] Philipp Bruland, Christian Forster, Bernhard Breil, Sonja Ständer, Martin Dugas und Fleur Fritz. „Does Single-Source create an added value? Evaluating the impact of introducing x4T into the clinical routine on workflow modifications, data quality and cost-benefit". Das Manuskript ist zur Veröffentlichung eingereicht. Jan. 2014.

[BFD12] Philipp Bruland, Christian Forster und Martin Dugas. „x4T-EDC: A Prototype for Study Documentation Based on the Single Source Concept". In: *24th International Conference of the European Federation for Medical Informatics Quality of Life through Quality of Information – J. Mantaset al. (Eds.) MIE2012 / CD / Short Communications (Oral)*. 2012.

[BG08] Eric Belden und Janis Greenberg. *Oracle Database Object-Relational Developer's Guide 11g Release 1 (11.1)*. Oracle. Aug. 2008. URL: **http://docs.oracle.com/cd/B28359_01/appdev.111/b28371.pdf**.

[BGD97] Andreas Behm, Andreas Geppert und Klaus R. Dittrich. „On the Migration of Relational Schemas and Data to Object-Oriented Database Systems". In: *In Proc. 5th International Conference on Re-Technologies for Information Systems*. Klagenfurt, Österreich, Dez. 1997, S. 13–33.

[BHB09] Alexander Benlian, Thomas Hess und Peter Buxmann. „Treiber der Adoption SaaS-basierter Anwendungen". German. In: *WIRTSCHAFTSINFORMATIK* 51.5 (2009), S. 414–428. DOI: **10.1007/s11576-009-0189-3**.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris und David Orchard. *Web Services Architecture*. W3C Working Group Note. W3C, Feb. 2004. URL: **http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/**.

[BJB09] Vasilis Boucharas, Slinger Jansen und Sjaak Brinkkemper. „Formalizing software ecosystem modeling". In: *Proceedings of the 1st international workshop on Open component ecosystems*. IWOCE '09. Amsterdam, Niederlande: ACM, 2009, S. 41–50. DOI: **10.1145/1595800.1595807**.

[BKK03] Martin Bernauer, Gerti Kappel und Gerhard Kramler. *Representing XML Schema in Uml - An UML Profile for XML Schema*. Techn. Ber. Vienna University of Technology, 2003.

[BKK04] Martin Bernauer, Gerti Kappel und Gerhard Kramler. „Representing XML Schema in UML – A Comparison of Approaches". In: *Web Engineering*. Hrsg. von Nora Koch, Piero Fraternali und Martin Wirsing. Bd. 3140. Lecture Notes in Computer Science. Berlin: Springer, 2004, S. 440–444. DOI: **10.1007/978-3-540-27834-4_54**.

[BKR12] Jörg Becker, Martin Kugeler und Michael Rosemann, Hrsg. *Prozessmanagement: Ein Leitfaden zur prozessorientierten Organisationsgestaltung*. Berlin: Springer Gabler, 2012.

[BL12] Martine Brachet und Ludy Limburg. „Moving High Value Payments into ISO 20022 – A Pragmatic Approach". In: *ISO 20022 Newsletter* 4.2 (2012), S. 5–6. URL: **http://www.iso20022.org/documents/general/ISO_20022_RMG_Newsletter_Fall_2012.pdf**.

[Boy09] John M. Boyer. *XForms 1.1*. W3C Recommendation. W3C, Okt. 2009. URL: **http://www.w3.org/TR/2009/REC-xforms-20091020/**.

[BPS+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler und François Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. W3C, Nov. 2008. URL: **http://www.w3.org/TR/2008/REC-xml-20081126/**.

[BPV12] Jörg Becker, Wolfgang Probandt und Oliver Vering. *Grundsätze ordnungsmäßiger Modellierung: Konzeption und Praxisbeispiel für ein effizientes Prozessmanagement*. BPM kompetent. Berlin: Springer, 2012.

[Bro08] G. Brooks. „Team Pace Keeping Build Times Down“. In: *Agile, 2008. AGILE '08. Conference*. 2008, S. 294–297. DOI: **10.1109/Agile.2008.41**.

[BS04] Jörg Becker und Reinhard Schütte. *Handelsinformationssysteme*. Redline Wirtschaft. Landsberg/Lech: verlag moderne industrie, 2004.

[BS98] D. Barry und T. Stanienda. „Solving the Java object storage problem“. In: *IEEE Computer* 31.11 (1998), S. 33–40. DOI: **10.1109/2.730734**.

[BSBG12] F. Belqasmi, J. Singh, S.Y. Bani Melhem und R.H. Glitho. „SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing“. In: *Internet Computing, IEEE* 16.4 (2012), S. 54–63. DOI: **10.1109/MIC.2012.62**.

[BSM+11] Bernhard Breil, Axel Semjonow, Carsten Müller-Tidow, Fleur Fritz und Martin Dugas. „HIS-based Kaplan-Meier plots–a single source approach for documenting and reusing routine survival information.“ eng. In: *BMC Medical Informatics and Decision Making* 11 (Feb. 2011), S. 11. DOI: **10.1186/1472-6947-11-11**.

[BW08] Wolf-Gideon Bleek und Henning Wolf. *Agile Softwareentwicklung*. Heidelberg: dpunkt.verlag, 2008.

[CBK10] William Candillon, Matthias Brantner und Dennis Knochenwefel. „XQuery Design Patterns“. In: *Proceedings of Balisage: The Markup Conference 2010*. Bd. 5. Balisage Series on Markup Technologies. Montréal, Kanada, 2010.

[CC05] Luca Cabibbo und Antonio Carosi. „Managing Inheritance Hierarchies in Object/Relational Mapping Tools“. In: *Advanced Information Systems Engineering*. Hrsg. von Oscar Pastor und João Falcão e Cunha. Bd. 3520. Lecture Notes in Computer Science. Berlin: Springer, 2005, S. 135–150. DOI: **10.1007/11431855_11**.

[CCN+99] Michael J. Carey, Donald D. Chamberlin, Srinivasa Narayanan, Bennet Vance, Doug Doole, Serge Rielau, Richard Swagerman und Nelson Mendonça Mattos. „O-O, What Have They Done to DB2?“ In: *Proceedings of the 25th International Conference on Very Large Data Bases*. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, S. 542–553. URL: **http://dl.acm.org/citation.cfm?id=645925.671522**.

[CD96] Michael J. Carey und David J. DeWitt. „Of Objects and Databases: A Decade of Turmoil“. In: *Proceedings of the 22th International Conference on Very Large Data Bases*. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, S. 3–14. URL: **http://dl.acm.org/citation.cfm?id=645922.673465**.

[CDI10] CDISC. *Operational Data Model*. Standard 1.3.1. Clinical Data Interchange Standards Consortium, Feb. 2010. URL: **http://www.cdisc.org/odm**.

[CFQ07] Juan Pablo Carvallo, Xavier Franch und Carme Quer. „Determining Criteria for Selecting Software Components: Lessons Learned". In: *Software, IEEE* 24.3 (2007), S. 84–94. DOI: **10.1109/MS.2007.70**.

[Che76] Peter Pin-Shan Chen. „The entity-relationship model – toward a unified view of data". In: *ACM Trans. Database Syst.* 1.1 (März 1976), S. 9–36. DOI: **10.1145/320434.320440**.

[CHL11] Swee-Mei Chin, Su-Cheng Haw und Chien-Sing Lee. „X-CM: A Conceptual Modeling for XML Databases". In: *2011 International Conference on Telecommunication Technology and Applications*. Bd. 5. Singapur: IACSIT Press, 2011, S. 59–63.

[CI05] William R. Cook und Ali H. Ibrahim. *Integrating Programming Languages & Databases: What's the Problem?* ODBMS.ORG, Expert Article. 2005. URL: **http://odbms.org/experts.aspx#article10**.

[CK02] Manuel M. T. Chakravarty und Gabriele C. Keller. *An Introduction to Computing with Haskell*. Pearson SprintPrint, 2002.

[CM84] George Copeland und David Maier. „Making smalltalk a database system". In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. SIGMOD '84. Boston, MA, USA: ACM, 1984, S. 316–325. DOI: **10.1145/602259.602300**.

[CMRW07] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman und Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation. W3C, Juni 2007. URL: **http://www.w3.org/TR/2007/REC-wsdl20-20070626**.

[CO95] Philip R. Cohen und Sharon L. Oviatt. „The role of voice input for human-machine communication". In: *Proceedings of the National Academy of Sciences* 92.22 (Okt. 1995), S. 9921–9927.

[Cod70] E. F. Codd. „A relational model of data for large shared data banks". In: *Commun. ACM* 13.6 (Juni 1970), S. 377–387. DOI: **10.1145/362384.362685**.

[Cod71] E. F. Codd. „Further Normalization of the Data Base Relational Model". In: *Data Base Systems*. Hrsg. von R. Rustin. Englewood Cliffs, NJ, USA: Prentice Hall, 1971, S. 33–64.

[Com79] Douglas Comer. „Ubiquitous B-Tree". In: *ACM Comput. Surv.* 11.2 (Juni 1979), S. 121–137. DOI: **10.1145/356770.356776**.

[CSF00] Rainer Conrad, Dieter Scheffner und J. Christoph Freytag. „XML conceptual modeling using UML". In: *Proceedings of the 19th international conference on Conceptual modeling*. ER'00. Salt Lake City, UT, USA: Springer-Verlag, 2000, S. 558–574. URL: **http://dl.acm.org/citation.cfm?id=1765112.1765165**.

[CVZ+02] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassilis J. Tsotras und Carlo Zaniolo. „Efficient structural joins on indexed XML documents“. In: *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB '02. Hong Kong, China: VLDB Endowment, 2002, S. 263–274. URL: **http://dl.acm.org/citation.cfm?id=1287369.1287393**.

[DAB+05] Robert H. Dolin, Liora Alschuler, Sandy Boyer, Calvin Beebe, Fred M. Behlen, Paul V. Biron und Amnon Shabo. *HL7 Clinical Document Architecture, Release 2.0*. Standard R2-2005. Ann Arbor, MI, USA: Health Level Seven Inc., Sep. 2005.

[Dau03] B. Daum. *Modeling Business Objects With XML Schema*. The Morgan Kaufmann Series in Software Engineering and Programming Series. Morgan Kaufmann, Apr. 2003. URL: **San%20Francisco,%20CA,%20USA**.

[DDO08] Remco M. Dijkman, Marlon Dumas und Chun Ouyang. „Semantics and analysis of business process models in BPMN“. In: *Inf. Softw. Technol.* 50.12 (Nov. 2008), S. 1281–1294. DOI: **10.1016/j.infsof.2008.02.006**.

[DFB+11] Philipp Dziuballe, Christian Forster, Bernhard Breil, Volker Thiemann, Fleur Fritz, Jens Lechtenbörger, Gottfried Vossen und Martin Dugas. „The Single Source Architecture x4T to Connect Medical Documentation and Clinical Research“. In: *User Centred Networked Health Care*. Hrsg. von Anne Moen, Stig Kjær Andersen, Jos Aarts und Petter Hurlen. Bd. 169. Studies in Health Technology and Informatics. iOS Press, 2011, S. 902–906.

[Dij68] E.W. Dijkstra. „A constructive approach to the problem of program correctness“. English. In: *BIT Numerical Mathematics* 8.3 (1968), S. 174–186. DOI: **10.1007/BF01933419**.

[Dij72] Edsger W. Dijkstra. „The humble programmer“. In: *Commun. ACM* 15.10 (Okt. 1972), S. 859–866. DOI: **10.1145/355604.361591**.

[Din13] Fabrizio Dinacci. „Migrating an RTGS system to ISO 20022: The strategy for TARGET2“. In: *ISO 20022 Newsletter* 5.2 (2013), S. 8–9. URL: **http://www.iso20022.org/documents/general/ISO_20022_RMG_Newsletter_Fall_2013.pdf**.

[DJMZ05] Wolfgang Dostal, Mario Jeckle, Ingo Melzer und Barbara Zengler. *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. München: Spektrum Akademischer Verlag, 2005.

[DKK07] Neil Daswani, Christoph Kern und Anita Kesavan. *Foundations of Security: What Every Programmer Needs to Know*. Berkeley, CA, USA: Apress, 2007.

[DMG07] Paul Duvall, Stephen M. Matyas und Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. The Addison-Wesley Signature Series. Boston, MA: Addison-Wesley Professional, 2007.

[Don06] Kevin Donnelly. „SNOMED-CT: The Advanced Terminology and Coding System for eHealth“. In: *Medical and Care Compunetics 3*. Hrsg. von L. Bos, L. Roa, K. Yogesan, B. O'Connell und A. Marsh. Bd. 121. Studies in Health Technology and Informatics. 2006.

[Dra13] Mark D Drake. *Oracle XML DB in Oracle Database 12c*. Oracle Corporation. Redwood Shores, CA, USA, Juni 2013. URL: **http://www.oracle.com/technetwork/database-features/xmldb/overview/xmldb-twp-12cr1-1964803.pdf**.

[Dub03] Micah Dubinko. *XForms Essentials*. Essentials Series. Sebastopol, CA, USA: O'Reilly Media, Aug. 2003.

[Ecm11] Ecma International. *Standard ECMA-262: ECMAScript® Language Specification*. Standard 5.1. Ecma International, Juni 2011. URL: **http://www.ecma-international.org/ecma-262/5.1/**.

[EG05] Anja Ebersbach und Markus Glaser. „Wiki“. German. In: *Informatik-Spektrum* 28.2 (2005), S. 131–135. DOI: **10.1007/s00287-005-0480-7**.

[Eis13] Andrew Eisenberg. „XQuery 3.0 is Nearing Completion“. In: *SIGMOD Rec.* 42.3 (Okt. 2013), S. 34–41. DOI: **10.1145/2536669.2536675**.

[EM04] Andrew Eisenberg und Jim Melton. „Advancements in SQL/XML“. In: *SIGMOD Rec.* 33.3 (Sep. 2004), S. 79–86. DOI: **10.1145/1031570.1031588**.

[EMK+04] Andrew Eisenberg, Jim Melton, Krishna Kulkarni, Jan-Eike Michels und Fred Zemke. „SQL:2003 has been published“. In: *SIGMOD Rec.* 33.1 (März 2004), S. 119–126. DOI: **10.1145/974121.974142**.

[EN02] Ramez Elmasri und Shamkant B. Navathe. *Grundlagen von Datenbanksystemen*. München: Pearson Education, 2002.

[Eng08] Robert A. Van Engelen. „A framework for service-oriented computing with C and C++ Web service components“. In: *ACM Trans. Internet Technol.* 8.3 (Mai 2008), 12:1–12:25. DOI: **10.1145/1361186.1361188**.

[Ens78] P. H. Enslow. „What is a 'Distributed' Data Processing System?“ In: *Computer* 11.1 (Jan. 1978), S. 13–21. DOI: **10.1109/C-M.1978.217901**.

[Erl05] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall International, Aug. 2005.

[Eur02] European Medicines Agency. *ICH Topic E 6 (R1) Guideline for Good Clinical Practice*. Juli 2002. URL: **http://www.ema.europa.eu/docs/en_GB/document_library/Scientific_guideline/2009/09/WC500002874.pdf**.

[FBL+14] Christian Forster, Philipp Bruland, Jens Lechtenbörger, Bernhard Breil und Gottfried Vossen. „Single-Source with x4T: Process Design, Architecture, and Use Cases“. Das Manuskript ist zur Veröffentlichung eingereicht. Jan. 2014.

[FC03] X. Franch und J.P. Carvallo. „Using quality models in software package selection". In: *Software, IEEE* 20.1 (2003), S. 34–41. DOI: **10.1109/MS.2003.1159027**.

[FF03] Deborah A. Fritz und Richard J. Fritz. *MARC21 for Everyone - A Practical Guide*. Chicago, IL, USA: American Library Association, 2003.

[Fie00] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures". Doctoral dissertation. Irvine, CA, USA: University of California, 2000.

[Fis12] Guido Fischermanns. *Praxishandbuch Prozessmanagement*. Bd. 9. ibo Schriftenreihe. Gießen: Verlag Dr. Götz Schmidt, 2012.

[FK92] David F. Ferraiolo und D. Richard Kuhn. „Role-Based Access Controls". In: *Proceedings of 15th National Computer Security Conference*. Baltimore, MD, USA, 1992, S. 554–563.

[FMD+96] A. W. Forrey, C. J. McDonald, G. DeMoor, S. M. Huff, D. Leavelle, D. Leland, T. Fiers, L. Charles, B. Griffin, F. Stalling, A. Tullis, K. Hutchins und J. Baenziger. „Logical observation identifier names and codes (LOINC) database: a public use set of codes and names for electronic reporting of clinical laboratory test results". In: *Clinical Chemistry* 42.1 (Jan. 1996), S. 81–90. URL: **http://www.clinchem.org/content/42/1/81**.

[Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. The Addison-Wesley Signature Series. Boston, MA, USA: Addison-Wesley Longman, 2003.

[FPL13] FPL Global Technical Committee. *FIX Specification*. Specification 5.0 Service Pack 2 with 20131209 Errata. New York, NY, USA: FIX Protocol Ltd, Dez. 2013. URL: **http://www.fixtradingcommunity.org/pg/structure/tech-specs/fix-version/50-service-pack-2**.

[FpM14] FpML Working Groups. *Financial product Markup Language*. Recommendation 5.6. International Swaps und Derivatives Association, Inc, Jan. 2014. URL: **http://www.fpml.org/spec/fpml-5-6-5-rec-1/**.

[FR12] Jakob Freund und Bernd Rücker. *Praxishandbuch BPMN 2.0*. München: Carl Hanser Verlag, 2012.

[Fra99] Piero Fraternali. „Tools and approaches for developing data-intensive Web applications: a survey". In: *ACM Comput. Surv.* 31.3 (Sep. 1999), S. 227–263. DOI: **10.1145/331499.331502**.

[FRL+11] AbdenNaji El Fadly, Bastien Rance, Noël Lucas, Charles Mead, Gilles Chatellier, Pierre-Yves Lastic, Marie-Christine Jaulent und Christel Daniel. „Integrating clinical research with the Healthcare Enterprise: From the REUSE project to the EHR4CR platform". eng. In: *Journal of Biomedical Informatics* 44, Supplement 1 (Dez. 2011), S94–S102. DOI: **10.1016/j.jbi.2011.07.007**.

[Ful08] James R. Fuller. *Advancing with XQuery: Develop application idioms*. developerWorks. IBM Corporation, Sep. 2008. URL: **http://www.webcomposite.com/resource/pdf/x-advxquery-pdf.pdf**.

[Gad13] Andreas Gadatsch. *Grundkurs Geschäftsprozess-Management Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. Wiesbaden: Springer Vieweg, 2013.

[Gal02] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. Wiley Desktop Editions Series. New York, NY, USA: John Wiley & Sons, Inc, 2002.

[GG01] Mark Graves und Charles F. Goldfarb. *Designing XML Databases*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[GGL10] Reyes Gonzalez, Jose Gasco und Juan Llopis. „Information systems outsourcing reasons and risks: a new assessment". In: *Industrial Management & Data Systems* 110.2 (2010), S. 284–303. DOI: **10.1108/02635571011020359**.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis, IN, USA: Addison-Wesley, 1994.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar und Yves Lafon. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C Recommendation. W3C, Apr. 2007. URL: **http://www.w3.org/TR/2007/REC-soap12-part1-20070427/**.

[Gib06] Jeremy Gibbons. „Design Patterns as Higher-Order Datatype-Generic Programs". In: *Workshop on Generic Programming*. Hrsg. von Ralf Hinze. Sep. 2006. URL: **http://www.comlab.ox.ac.uk/jeremy.gibbons/publications/hodgp.pdf**.

[GL10] N. Gruschka und L. Lo Iacono. „Server-Side Streaming Processing of Secured MTOM Attachments". In: *Web Services (ECOWS), 2010 IEEE 8th European Conference on*. 2010, S. 11–18. DOI: **10.1109/ECOWS.2010.23**.

[GLQ11] Pierre Genevès, Nabil Layaïda und Vincent Quint. „Impact of XML Schema Evolution". In: *ACM Trans. Internet Technol.* 11.1 (Juli 2011), 4:1–4:27. DOI: **10.1145/1993083.1993087**.

[GM11] Clemens Gull und Stefan Münz. *HTML5 Handbuch*. Haar bei München: Franzis Verlag, 2011.

[Gol11] Joachim Goll. *Methoden und Architekturen der Softwaretechnik*. Wiesbaden: Vieweg+Teubner Verlag, 2011.

[Gol91] Charles F. Goldfarb. *The SGML Handbook*. Hrsg. von Yuri Rubinsky. Oxford, UK: Oxford University Press, 1991.

[GST12] Shudi Gao, C. M. Sperberg-McQueen und Henry S. Thompson. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C Recommendation. W3C, Apr. 2012. URL: **http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/**.

[GSW+00] J.-M. Le Goff, H. Stockinger, I. Willers, R. McClatchey, Z. Kovacs, P. Martin, N. Bhatti und W. Hassan. „Object Serialization and Deserialization Using XML". In: *Advances in Data Management 2000*. Hrsg. von Krithi Ramamritham und T.M. Vijayaraman. Tata McGraw-Hill Publishing Company, 2000.

[Ham04] Beda Christoph Hammerschmidt. „KeyX: ein selektiver schlüsselorientierter Index für das Index Selection Problem in XDBMS". In: *16. Workshop über Grundlagen von Datenbanken GI-Arbeitskreis "Grundlagen von Informationssystemen"*. Hrsg. von Stefan Conrad Mireille Samia. Monheim, Juni 2004, S. 63–67. URL: **http://www.dbs.cs.uni-duesseldorf.de/gvd2004/papers/HammerschmidtBedaChristoph.pdf**.

[Has00] Wilhelm Hasselbring. „Information system integration". In: *Commun. ACM* 43.6 (Juni 2000), S. 32–38. DOI: **10.1145/336460.336472**.

[Has12] Till Haselmann. „Cloud-Services in kleinen und mittleren Unternehmen: Nutzen, Vorgehen, Kosten". Dissertation. Westfälische Wilhelms-Universität Münster, Mai 2012.

[HB04] Hugo Haas und Allen Brown. *Web Services Glossary*. W3C Working Group Note. W3C, Feb. 2004. URL: **http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/**.

[HFT12] Keman Huang, Yushun Fan und Wei Tan. „An Empirical Study of Programmable Web: A Network Analysis on a Service-Mashup System". In: *Web Services (ICWS), 2012 IEEE 19th International Conference on*. 2012, S. 552–559. DOI: **10.1109/ICWS.2012.32**.

[HHZ11] Bernd Heinrich, Andreas Huber und Steffen Zimmermann. „Make-and-Sell or Buy of Web Services - A Real Option Approach". In: *Proceedings of the 19th European Conference on Information Systems (ECIS)*. Hrsg. von Virpi Kristiina Tuunainen, Matti Rossi und Joe Nandhakumar. Helsinki, Finland, Juni 2011.

[Hin07] Andrew Hinchley. *Understanding Version 3: A primer on the HL7 Version 3 Healthcare Interoperability Standard - Normative Edition*. München: Alexander Mönch Publishing, 2007.

[HJR+03] Marjan Hericko, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec und Ales Zivkovic. „Object serialization analysis and comparison in Java and .NET". In: *SIGPLAN Not.* 38.8 (Aug. 2003), S. 44–54. DOI: **10.1145/944579.944589**.

[HKKR05] Martin Hitz, Gerti Kappel, Elisabeth Kapsammer und Werner Retschitzegger. *UML@Work*. Heidelberg: dpunkt.verlag, 2005.

[HLV07] Stephan Hagemann, Carolin Letz und Gottfried Vossen. „Web Service Discovery - Reality Check 2.0“. In: *Proceedings of the Third International Conference on Next Generation Web Services Practices*. NWESP ’07. Washington, DC, USA: IEEE Computer Society, 2007, S. 113–118. DOI: **10.1109/NWESP.2007.31**.

[Hoa69] C. A. R. Hoare. „An axiomatic basis for computer programming“. In: *Commun. ACM* 12.10 (Okt. 1969), S. 576–580. DOI: **10.1145/363235.363259**.

[Hoe11] Robert Hoekman. *Designing the Obvious: A Common Sense Approach to Web Application Design*. Berkeley, CA, USA: New Riders, 2011.

[Hog11] Brian P. Hogan. *HTML5 and CSS3: Develop with Tomorrow’s Standards Today*. Pragmatic Programmers, 2011.

[Hoh96] Uwe Hohenstein. „Bridging the gap between C++ and relational databases“. In: *ECOOP ’96 — Object-Oriented Programming*. Hrsg. von Pierre Cointe. Bd. 1098. Lecture Notes in Computer Science. Berlin: Springer, 1996, S. 398–420. DOI: **10.1007/BFb0053071**.

[HP06] Mikko Honkala und Mikko Pohja. „Multimodal Interaction with XForms“. In: *Proceedings of the 6th international Conference on Web Engineering*. ICWE ’06. Palo Alto, CA, USA: ACM, 2006, S. 201–208. DOI: **10.1145/1145581.1145624**.

[HT99] Andrew Hunt und David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Reading, MA, USA: Addison-Wesley Longman, Inc., Okt. 1999.

[IBM13] IBM Corporation. *DB2 11 for z/OS: pureXML Guide*. Dez. 2013. URL: **http://publib.boulder.ibm.com/epubs/pdf/dsnxgn01.pdf**.

[IBNW09] C. Ireland, D. Bowers, M. Newton und K. Waugh. „A Classification of Object-Relational Impedance Mismatch“. In: *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA ’09. First International Conference on*. 2009, S. 36–43. DOI: **10.1109/DBKDA.2009.11**.

[IBNW11] Christopher Ireland, David Bowers, Michael Newton und Kevin Waugh. „Exploring the essence of an Object-Relational Impedance Mismatch: a novel technique based on equivalence in the context of a framework“. In: *DBKDA 2011, The Third International Conference on Advances in Databases, Knowledge, and Data Applications*. Jan. 2011, S. 65–70. URL: **http://oro.open.ac.uk/30219/**.

[ISO13] ISO 20022. *Financial Services - Universal financial industry message scheme*. Techn. Ber. 20022. Genf, Schweiz: International Organization for Standardization, ISO, Mai 2013.

[Jac05] Ian Jacobs. *World Wide Web Consortium Process Document*. Techn. Ber. W3C, Okt. 2005. URL: **http://www.w3.org/2005/10/Process-20051014/**.

[JD03] Jeffrey N. Johnson und Paul F. Dubois. „Issue Tracking“. In: *Computing in Science and Engg.* 5.6 (Nov. 2003), S. 71–77. DOI: **10.1109/MCISE.2003.1238707**.

[Jen87] Kurt Jensen. „Coloured Petri nets“. In: *Petri Nets: Central Models and Their Properties*. Hrsg. von W. Brauer, W. Reisig und G. Rozenberg. Bd. 254. Lecture Notes in Computer Science. Berlin: Springer, 1987, S. 248–299. DOI: **10.1007/BFb0046842**.

[Jit11] Jitendra Kotamraju (Maintenance Lead). *Java API for XML-Based Web Services 2.2 Rev a*. Java Specification Request 224. Oracle America, Inc., Dez. 2011. URL: **http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index4.html**.

[Joh07] Jeff Johnson. *GUI Bloopers 2.0: Common User Interface Design Don'ts and Dos*. Interactive Technologies. Burlington, MA, USA: Morgan Kaufmann, 2007.

[Jos08] Nicolai Josuttis. *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*. Heidelberg: dpunkt.verlag, 2008.

[JPMM04] Stefan Jablonski, Ilia Petrov, Christian Meiler und Udo Mayer. *Guide to Web Application and Platform Architectures*. Berlin: Springer, 2004.

[KAR+07] Rebecca Kush, Liora Alschuler, Roberto Ruggeri, Sally Cassells, Nitin Gupta, Landen Bain, Karen Claise, Monica Shah und Meredith Nahm. „Implementing Single Source: the STARBRITE proof-of-concept study“. In: *Journal of the American Medical Informatics Association* 14.5 (2007), S. 662–673. DOI: **10.1197/jamia.M2157**.

[Kar12] Thomas Karle. „Kollaborative Softwareentwicklung auf Basis serviceorientierter Architekturen“. Dissertation. Karlsruhe: Karlsruher Institut für Technologie (KIT), 2012.

[Kay05] Michael Kay. „Comparing XSLT and XQuery“. In: *XTech 2005 Conference Proceedings*. Amsterdam, Niederlande, Mai 2005. URL: **http://www.mscs.mu.edu/~praveen/Teaching/fa05/AdvDb/PaperTeams/XSLT2XQTeam/Comparing%20XSLT%20and%20XQuery.htm**.

[Kay07] Michael Kay. *XSL Transformations (XSLT) Version 2.0*. W3C Recommendation. W3C, Jan. 2007. URL: **http://www.w3.org/TR/2007/REC-xslt20-20070123/**.

[Kay08] Michael Kay. *XSLT 2.0 and XPath 2.0*. Indianapolis, IN, USA: Wiley Publishing, Inc., 2008.

[Kay13] Michael Kay. *XPath and XQuery Functions and Operators 3.0*. W3C Candidate Recommendation. W3C, Jan. 2013. URL: **http://www.w3.org/TR/xpath-functions-30/**.

[KBB10] Philip Kotler, Roland Berger und Nils Bickhoff. *The Quintessence of Strategic Management: What You Really Need to Know to Survive in Business*. Berlin: Springer-Verlag, 2010. URL: **http://www.amazon.com/The-Quintessence-Strategic-Management-Business-ebook/dp/B008PP9DZ0?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B008PP9DZ0**.

[KCW10] D. Richard Kuhn, Edward J. Coyne und Timothy R. Weil. „Adding Attributes to Role-Based Access Control". In: *IEEE Computer* 43.6 (2010), S. 79–81. DOI: **10.1109/MC.2010.155**.

[Kep04] Stephan Kepser. „A Simple Proof of the Turing-Completeness of XSLT and XQuery." In: *Proceedings of the Extreme Markup Languages 2004 Conference*. Hrsg. von Tommie Usdi. Montréal, QC, Kanada, Aug. 2004.

[Keß08] Mirjam Keßler. „KIM – Kompetenzzentrum Interoperable Metadaten". In: *Dialog mit Bibliotheken* 1 (2008), S. 22–24.

[KJA93] Arthur M. Keller, Richard Jensen und Shailesh Agarwal. „Persistence software: bridging object-oriented programming and relational databases". In: *SIGMOD Rec.* 22.2 (Juni 1993), S. 523–528. DOI: **10.1145/170036.171541**.

[KKLM10] Jakub Klímek, Lukáš Kopenec, Pavel Loupal und Jakub Malý. „XCase - A Tool for Conceptual XML Data Modeling". In: *Advances in Databases and Information Systems*. Hrsg. von Janis Grundspenkis, Marite Kirikova, Yannis Manolopoulos und Leonids Novickis. Bd. 5968. Lecture Notes in Computer Science. Berlin: Springer, 2010, S. 96–103. DOI: **10.1007/978-3-642-12082-4_13**.

[KKS+13] Felix Köpcke, Stefan Kraus, Axel Scholler, Carla Nau, Jürgen Schüttler, Hans-Ulrich Prokosch und Thomas Ganslandt. „Secondary use of routinely collected patient data in a clinical trial: An evaluation of the effects on patient recruitment and data acquisition". In: *International Journal of Medical Informatics* 82.3 (März 2013), S. 185–192. DOI: **10.1016/j.ijmedinf.2012.11.008**.

[KL06] Solmaz Kolahi und Leonid Libkin. „On redundancy vs dependency preservation in normalization: an information-theoretic study of 3NF". In: *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. PODS '06. Chicago, IL, USA: ACM, 2006, S. 114–123. DOI: **10.1145/1142351.1142369**.

[KL96] Wilhelm Kamlah und Paul Lorenzen. *Logische Propädeutik: Vorschule des vernünftigen Redens*. Stuttgart: Metzler, 1996.

[Knu76] Donald E. Knuth. „Big Omicron and Big Omega and Big Theta". In: *SIGACT News* 8.2 (Apr. 1976), S. 18–24. DOI: **10.1145/1008328.1008329**.

[KÖD11] Patrick Kling, M. Tamer Özsu und Khuzaima Daudjee. „Scaling XML query processing: distribution, localization and pruning". In: *Distributed and Parallel Databases* 29.5-6 (2011), S. 445–490. DOI: **10.1007/s10619-011-7085-8**.

[Koh09] Kohsuke Kawaguchi (Maintenance Lead). *Java Architecture for XML Binding (JAXB) 2.2*. Java Specification Request 224. Oracle America, Inc., Dez. 2009. URL: **http://jcp.org/aboutJava/communityprocess/mrel/jsr222/index2.html**.

[Kop08] Mikael Kopteff. „The Usage and Performance of Object Databases compared with ORM tools in a Java environment". In: *International Conference on Object Databases*. Berlin, März 2008. URL: **http://www.odbms.org/download/045.01%20Kopteff%20The%20Usage%20and%20Performance%20of%20Object%20Databases%20Compared%20with%20ORM%20Tools%20in%20a%20Java%20Environment%20March%202008.PDF**.

[KOY+10] Wolfgang Kuchinke, Christian Ohmann, Qin Yang u. a. „Heterogeneity prevails: the state of clinical trial data management in Europe - results of a survey of ECRIN centres." eng. In: *Trials* 11 (2010), S. 79. DOI: **10.1186/1745-6215-11-79**.

[KP88] Glenn E. Krasner und Stephen T. Pope. „A cookbook for using the model-view controller user interface paradigm in Smalltalk-80". In: *J. Object Oriented Program.* 1.3 (Aug. 1988), S. 26–49. URL: **http://dl.acm.org/citation.cfm?id=50757.50759**.

[Kru06] Steve Krug. *Don't make me think! Web Usability – Das intuitive Web*. Heidelberg: mitp, 2006.

[KS09] Jane Kaye und Mark Stranger, Hrsg. *Principles and Practice in Biobank Governance*. Farnham, Surrey, UK: Ashgate Publishing Limited, 2009.

[KS91] Frank A. Koch und Peter Schnupp. *Software-Recht: Band 1*. Hrsg. von M. Nagl, P. Schnupp und H. Strunz. Berlin: Springer, 1991.

[KTM+99] Barbara A. Kitchenham, Guilherme H. Travassos, Anneliese von Mayrhauser, Frank Niessink, Norman F. Schneidewind, Janice Singer, Shingo Takada, Risto Vehvilainen und Hongji Yang. „Towards an Ontology of software maintenance". In: *Journal of Software Maintenance* 11.6 (Nov. 1999), S. 365–389. DOI: **10.1002/(SICI)1096-908X(199911/12)11:6<365::AID-SMR200>3.0.CO;2-W**.

[Kug00] Martin Kugeler. *Informationsmodellbasierte Organisationsgestaltung: Modellierungskonventionen und Referenzvorgehensmodell zur prozessorientierten Reorganisation*. Berlin: Logos Verlag, 2000.

[KVKV12] Lennart C.L. Kats, Richard G. Vogelij, Karl Trygve Kalleberg und Eelco Visser. „Software Development Environments on the Web: A Research Agenda". In: *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. Onward! '12. Tucson, AZ, USA: ACM, 2012, S. 99–116. DOI: **10.1145/2384592.2384603**.

[Lau04] Andrew M. St. Laurent. *Understanding Open Source & Free Software Licensing*. Hrsg. von Simon St. Laurent. Sebastopol, CA, USA: O'Reilly Media, Aug. 2004.

[LC62] J. C. R. Licklider und Welden E. Clark. „On-line man-computer communication“. In: *Proceedings of the May 1-3, 1962, spring joint computer conference*. AIEE-IRE ’62 (Spring). San Francisco, CA, USA: ACM, 1962, S. 113–128. DOI: **10.1145/1460833.1460847**.

[LD04] P. C. Lockemann und K. R. Dittrich. *Architektur von Datenbanksystemen*. Heidelberg: dpunkt.verlag, 2004.

[LDG12] Simon St. Laurent, Edd Dumbill und Eric J. Gruber. *Learning Rails 3*. Sebastopol, CA, USA: O’Reilly Media, Juli 2012.

[LH08] James P. Lawler und H. Howell-Barber. *Service-oriented architecture: SOA strategy, methodology, and technology*. Boca Raton, FL, USA: Auerbach Publications, 2008.

[LKN+06] Kelvin Lawrence, Chris Kaler, Anthony Nadalin, Ronald Monzillo und Phillip Hallam-Baker. *Web Services Security: SOAP Message Security 1.1*. OASIS Standard Specification. OASIS, Feb. 2006. URL: **http://docs.oasis-open.org/wss/v1.1/**.

[LLY06] HongXing Liu, YanSheng Lu und Qing Yang. „XML Conceptual Modeling with XUML“. In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE ’06. Shanghai, China: ACM, 2006, S. 973–976. DOI: **10.1145/1134285.1134466**.

[LO01] Kirsten Lenz und Andreas Oberweis. „Modeling Interorganizational Workflows with XML Nets“. In: *Proceedings of the Hawai’i International Conference On System Sciences*. IEEE. Maui,HI, Jan. 2001. DOI: **10.1109/HICSS.2001.927083**.

[LO02] Kirsten Lenz und Andreas Oberweis. „Integrierte Dokumenten- und Ablaufmodellierung von E-Business-Prozessen“. In: *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen - Promise*. Hrsg. von Mathias Weske Jörg Desel. Bd. 21. LNI. Potsdam: GI, Okt. 2002, S. 40–51. URL: **http://subs.emis.de/LNI/Proceedings/Proceedings21/article555.html**.

[LO03] Kirsten Lenz und Andreas Oberweis. „Interorganizational Business Process Management with XML Nets“. In: *Petri Net Technology for Communication-Based Systems, Advances in Petri Nets*. Hrsg. von H. Ehrig, W. Reisig, G. Rozenberg und H. Weber. Bd. 2472. LNCS. Springer-Verlag, 2003, S. 243–263.

[Lou06] P. Louridas. „Using wikis in software development“. In: *Software, IEEE* 23.2 (2006), S. 88–91. DOI: **10.1109/MS.2006.62**.

[Lov02] Doug Lovell. *XSL Formatting Objects Developer’s Handbook*. Indianapolis, IN, USA: Sams Publishing, 2002.

[LOZ11] Yu Li, Andreas Oberweis und Huayu Zhang. „An integrated approach for modeling and facilitating RFID-based collaborative logistics processes“. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. SAC '11. TaiChung, Taiwan: ACM, 2011, S. 301–307. DOI: **10.1145/1982185.1982253**.

[LR13] Gibson Lam und David Rossiter. „A Web Service Framework Supporting Multimedia Streaming“. In: *Services Computing, IEEE Transactions on* 6.3 (2013), S. 400–413. DOI: **10.1109/TSC.2012.11**.

[LS04] Wolfgang Lehner und Harald Schöning. *XQuery: Grundlagen und fortgeschrittene Methoden*. Heidelberg: dpunkt.verlag, 2004.

[LS80] Bennett P. Lientz und E. Burton Swanson. *Software Maintenance Management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980.

[LSLV11] Markku Laine, Denis Shestakov, Evgenia Litvinova und Petri Vuorimaa. „Toward Unified Web Application Development“. In: *IT Professional* 13.5 (2011), S. 30–36. DOI: **10.1109/MITP.2011.55**.

[LSR03] Bernadette Farias Lóscio, Ana Carolina Salgado und Luciano do Rêgo Galvão. „Conceptual modeling of XML schemas“. In: *Proceedings of the 5th ACM international workshop on Web information and data management*. WIDM '03. New Orleans, LA, USA: ACM, 2003, S. 102–105. DOI: **10.1145/956699.956722**.

[LSSS07] M. Lühring, K. Sattler, E. Schallehn und K. Schmidt. „Autonomes Index Tuning – DBMS-integrierte Verwaltung von Soft Indexen“. In: *Datenbanksysteme in Business, Technologie und Web (BTW 2007), 12. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS), Proceedings, 7.-9. März 2007, Aachen*. Bd. 103. LNI. GI, 2007, S. 152–171.

[LSV12] Markku Laine, Denis Shestakov und Petri Vuorimaa. „XFormsDB: A Declarative Web Application Framework“. In: *Web Engineering*. Hrsg. von Marco Brambilla, Takehiro Tokuda und Robert Tolksdorf. Bd. 7387. Lecture Notes in Computer Science. Berlin: Springer, 2012, S. 477–480. URL: **http://dx.doi.org/10.1007/978-3-642-31753-8_48**.

[LW07] Einar Landre und Harald Wesenberg. „REST versus SOAP as Architectural Style for Web Services“. In: *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007*. Hrsg. von Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes und Guy L. Steele Jr. Okt. 2007.

[LW92] Barbara Liskov und Jeannette M. Wing. *Family Values: A Semantic Notion of Subtyping*. Techn. Ber. Pittsburgh, PA, USA, 1992.

[Maj92] Mila E. Majster-Cederbaum. „Ensuring the Existence of a BCNF-Decomposition that Preserves Functional Dependencies in $O(N^2)$ Time". In: *Inf. Process. Lett.* 43.2 (1992), S. 95–100.

[MB06] Jim Melton und Stephen Buxton. *Querying XML: XQuery, XPath, and SQL/XML in Context*. San Francisco, CA, USA: Morgan Kaufmann, 2006.

[MBK+11] Sebastian Mate, Thomas Bürkle, Felix Köpcke, Bernhard Breil, Bernd Wullich, Martin Dugas, Hans-Ulrich Prokosch und Thomas Ganslandt. „Populating the i2b2 Database with Heterogeneous EMR Data: a Semantic Network Approach". In: *User Centred Networked Health Care*. Hrsg. von Anne Moen, Stig Kjær Andersen, Jos Aarts und Petter Hurlen. Bd. 169. Studies in Health Technology and Informatics. iOS Press, 2011, S. 502–506. DOI: **10.3233/978-1-60750-806-9-502**.

[MC05] David Machin und Michael J. Campbell. *The Design of Studies for Medical Research*. Chichester, UK: John Willey & Sons Ltd, 2005.

[McB00] Pete McBreen. „Applying the Lessons of eXtreme Programming". In: *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'oo)*. TOOLS '00. Washington, DC, USA: IEEE Computer Society, 2000, S. 423–430. URL: **http://dl.acm.org/citation.cfm?id=832261.833308**.

[McB06] Darin McBeath. *XQuery Style Conventions*. Jan. 2006. URL: **http://xqdoc.org/xquery-style.pdf**.

[McC93] Steve McConnell. *Code complete: a practical handbook of software construction*. Code Series. Redmond, WA, USA: Microsoft Press, 1993.

[MD10] T. Mens und S. Demeyer, Hrsg. *Software Evolution*. Berlin: Springer, 2010.

[ME13] Karl-Heinz Menges und Expertenfachgruppe 11 „Computergestützte Systeme". *Überwachung computergestützter Systeme*. Aide-mémoire 07121202. Darmstadt: Zentralstelle der Länder für Gesundheitsschutz bei Arzneimitteln und Medizinprodukten, 2013. URL: **https://www.zlg.de/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/downloads/AM/QS/07121202.pdf&i_i=0&hash=e48d9bf7fa19237b8c3b035ae8126005401f6cc4**.

[ME92] Priti Mishra und Margaret H. Eich. „Join Processing in Relational Databases". In: *ACM Comput. Surv.* 24.1 (März 1992), S. 63–113. DOI: **10.1145/128762.128764**.

[Mei03] Wolfgang Meier. „eXist: An Open Source Native XML Database". In: *Web, Web-Services, and Database Systems*. Hrsg. von Akmal Chaudhri, Mario Jeckle, Erhard Rahm und Rainer Unland. Bd. 2593. Lecture Notes in Computer Science. Berlin: Springer, 2003, S. 169–183. DOI: **10.1007/3-540-36560-5_13**.

[Mel03] Jim Melton. *Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features*. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[Mel13] Jim Melton. *XQueryX 3.0*. W3C Candidate Recommendation. W3C, Jan. 2013. URL: **http://www.w3.org/TR/2013/CR-xqueryx-30-20130108/**.

[Mer03] Peter Mertens. „XML als Grundlage für standardisierte Internetschnittstellen". In: *XML-Komponenten in der Praxis*. Hrsg. von Peter Mertens. Xpert.press. Berlin: Springer, 2003, S. 9–27. DOI: **10.1007/978-3-642-55619-7_2**.

[Mes07] G. Meszaros. *XUnit Test Patterns: Refactoring Test Code*. The Addison-Wesley Signature Series. Boston, MA, USA: Addison Wesley Professional, 2007.

[MG09] Gavin Mulligan und Denis Gračanin. „A comparison of SOAP and REST implementations of a service based interaction independence middleware framework". In: *Winter Simulation Conference*. WSC '09. Austin, Texas: Winter Simulation Conference, 2009, S. 1423–1432. URL: **http://dl.acm.org/citation.cfm?id=1995456.1995650**.

[MIA10] Shane McCarron, Masayasu Ishikawa und Murray Altheim. *XHTML 1.1 - Module-based XHTML - Second Edition*. W3C Recommendation. W3C, Nov. 2010. URL: **http://www.w3.org/TR/2010/REC-xhtml11-20101123/**.

[Mil08] A. Miller. „A Hundred Days of Continuous Integration". In: *Agile, 2008. AGILE '08. Conference*. 2008, S. 289–293. DOI: **10.1109/Agile.2008.8**.

[ML86] Lothar F Mackert und Guy M Lohman. „R* optimizer validation and performance evaluation for local queries". In: *ACM SIGMOD Record*. Bd. 15. 2. ACM. 1986, S. 84–95.

[MLM+06] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown und Rebekah Metz. *Reference Model for Service Oriented Architecture 1.0*. Committee Specification 1. OASIS, Aug. 2006. URL: **http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf**.

[MML07] Mirella M. Moro, Susan Malaika und Lipyeow Lim. „Preserving XML Queries During Schema Evolution". In: *Proceedings of the 16th International Conference on World Wide Web*. WWW '07. Banff, Alberta, Canada: ACM, 2007, S. 1341–1342. DOI: **10.1145/1242572.1242841**.

[MNS05] Michael zur Muehlen, Jeffrey V. Nickerson und Keith D. Swenson. „Developing web services choreography standards—the case of REST vs. SOAP". In: *Decision Support Systems* 40.1 (2005), S. 9–29. DOI: **10.1016/j.dss.2004.04.008**.

[MO08] Hussein Morsy und Tanja Otto. *Ruby on Rails 2: Das Entwickler-Handbuch*. Bonn: Galileo Press, 2008.

[Mol09] Ian Molyneaux. *The Art of Application Performance Testing*. Sebastopol, CA, USA: O'Reilly Media, 2009.

[MPR03] Shane McCarron, Steven Pemberton und T. V. Raman. *XML Events: An Events Syntax for XML*. W3C Recommendation. W3C, Okt. 2003. URL: **http://www.w3.org/TR/2003/REC-xml-events-20031014/**.

[MS02] Jim Melton und Alan R. Simon. *SQL: 1999 - Understanding Relational Language Components*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2002.

[MS86] David Maier und Jacob Stein. „Indexing in an object-oriented DBMS". In: *Proceedings on the 1986 international workshop on Object-oriented database systems*. OODS '86. Pacific Grove, California, USA: IEEE Computer Society Press, 1986, S. 171–182. URL: **http://dl.acm.org/citation.cfm?id=318826.318850**.

[MSBT04] G.J. Myers, C. Sandler, T. Badgett und T.M. Thomas. *The Art of Software Testing*. Business Data Processing: A Wiley Series. Hoboken, NJ, USA: John Wiley & Sons, 2004.

[MSUW02] Stephen J. Mellor, Kendall Scott, Axel Uhl und Dirk Weise. „Model-Driven Architecture". English. In: *Advances in Object-Oriented Information Systems*. Hrsg. von Jean-Michel Bruel und Zohra Bellahsene. Bd. 2426. Lecture Notes in Computer Science. Berlin: Springer, 2002, S. 290–297. DOI: **10.1007/3-540-46105-1_33**.

[MT07] Tommi Mikkonen und Antero Taivalsaari. *Web Applications - Spaghetti Code for the 21st Century*. Techn. Ber. Mountain View, CA, USA: Sun Microsystems, 2007. URL: **http://dl.acm.org/citation.cfm?id=1698200**.

[MT79] Robert Morris und Ken Thompson. „Password security: A case history". In: *Communications of the ACM* 22.11 (Nov. 1979), S. 594–597. DOI: **10.1145/359168.359172**.

[Mur08] Peter Murray-Rust. „Open Data in Science". In: *Serials Review* 34.1 (2008), S. 52–64. DOI: **10.1016/j.serrev.2008.01.001**.

[Mur99] Vishu Krishnamurthy Muralidhar Subramanian. „Performance Challenges in Object-Relational DBMSs". In: *IEEE Data Eng. Bull.* 22.2 (Juni 1999), S. 27–31.

[Nar08] Philippe Narbel. „Object-Oriented Technology: A Multiparadigmatic Study of the Object-Oriented Design Patterns". In: *MPOOL'07, European Conference on Object-Oriented Programming*. Hrsg. von Michael Cebulla. Lecture Notes in Computer Science 4906. Berlin: Springer, 2008. URL: **http://deptinfo.labri.fr/~narbel/Pap/patMulti.pdf**.

[Nec07] Martin Necasky. „XSEM - A Conceptual Model for XML". In: *Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007)*. Hrsg. von John F. Roddick und Annika Hinze. Bd. 67. CRPIT. Ballarat, Australien: ACS, 2007, S. 37–48.

[NL05] Eric Newcomer und Greg Lomow. *Understanding SOA with Web Services*. Boston, MA, USA: Addison-Wesley Professional, 2005.

[Obj11] Object Management Group. *Business Process Model and Notation (BPMN) 2.0*. Spezifikation. Jan. 2011. URL: **http://www.omg.org/spec/BPMN/2.0**.

[ÖK10] M. Tamer Özsu und Patrick Kling. „Distributed XML Query Processing“. In: *Database and XML Technologies*. Hrsg. von MongLi Lee, JeffreyXu Yu, Zohra Bellahsène und Rainer Unland. Bd. 6309. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, S. 1–2. DOI: **10.1007/978-3-642-15684-7_1**.

[ONe08] Elizabeth J. O'Neil. „Object/relational mapping 2008: hibernate and the entity data model (edm)“. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, S. 1351–1356. DOI: **10.1145/1376616.1376773**.

[OS04] Nicola Onose und Jerome Simeon. „XQuery at your web service“. In: *Proceedings of the 13th international conference on World Wide Web*. WWW '04. New York, NY, USA: ACM, 2004, S. 603–611. DOI: **10.1145/988672.988754**.

[Ouc80] William G. Ouchi. „Markets, Bureaucracies, and Clans“. In: *Administrative Science Quarterly* 25.1 (1980), S. 129–141.

[Pap08] M. Papazoglou. *Web Services: Principles and Technology*. Harlow, UK: Pearson Education Ltd, 2008.

[Paw02] Dave Pawson. *XSL-FO*. Sebastopol, CA, USA: O'Reilly Media, Aug. 2002.

[PBG07] T. Posch, K. Birken und M. Gerdom. *Basiswissen Softwarearchitektur*. Heidelberg: dpunkt-Verlag, 2007.

[PDMP05] Torsten Priebe, Wolfgang Dobmeier, Björn Muschall und Günther Pernul. „ABAC - Ein Referenzmodell für attributbasierte Zugriffskontrolle“. In: *Tagungsband der 2. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik (Sicherheit 2005)*. 2005, S. 285–296.

[PDZ09] Szabolcs Payrits, Péter Dornbach und István Zólyomi. „XML Data Binding for C++ Using Metadata“. In: *International Journal of Web Services Research* 6.3 (2009), S. 18–34. DOI: **10.4018/jwsr.2009070102**.

[Per11a] Simon Perreault. *vCard Format Specification*. RFC 6350. Internet Engineering Task Force, Aug. 2011. URL: **http://www.ietf.org/rfc/rfc6350.txt**.

[Per11b] Simon Perreault. *xCard: vCard XML Representation*. RFC 6351. Internet Engineering Task Force, Aug. 2011. URL: **http://www.ietf.org/rfc/rfc6351.txt**.

[Pet62] Carl Adam Petri. „Kommunikation mit Automaten“. Dissertation. Technische Universität Darmstadt, 1962.

[Pet93] Margaret A. Peteraf. „The cornerstones of competitive advantage: A resource-based view“. In: *Strategic Management Journal* 14.3 (März 1993), S. 179–191.

[PFMP04] Torsten Priebe, Eduardo B. Fernandez, Jens I. Mehlau und Günther Pernul. „A Pattern System for Access Control“. In: *18th. Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. Sitges, Spanien: Kluwer, Juli 2004.

[PG09] H. U. Prokosch und T. Ganslandt. „Perspectives for medical informatics. Reusing the electronic medical record for clinical research". In: *Methods of Information in Medicine* 48.1 (2009), S. 38–44. DOI: **10.3414/ME9132**.

[PGM+12] David Peterson, Shudi Gao, Ashok Malhotra, C. M. Sperberg-McQueen und Henry S. Thompson. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. W3C Recommendation. W3C, Apr. 2012. URL: **http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/**.

[Pia83] Gregory Piatetsky-Shapiro. „The optimal selection of secondary indices is NP-complete". In: *SIGMOD Rec.* 13.2 (Jan. 1983), S. 72–75. DOI: **10.1145/984523.984530**.

[PIC07] PIC/S Secretariat. *Good Practices for Computerised Systems in Regulated "GxP" Environments*. Guidance document PI 011-3. Pharmaceutical Inspection Convention und Pharmaceutical Inspection Co-operation Scheme, Sep. 2007. URL: **http://www.picscheme.org/pdf/27_pi-011-3-recommendation-on-computerised-systems.pdf**.

[PSH97] V.I. Pavlovic, R. Sharma und T.S. Huang. „Visual interpretation of hand gestures for human-computer interaction: a review". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.7 (1997), S. 677–695. DOI: **10.1109/34.598226**.

[PY02] Mike P. Papazoglou und Jian Yang. „Design Methodology for Web Services and Business Processes". In: *Technologies for E-Services*. Hrsg. von Alejandro Buchmann, Ludger Fiege, Fabio Casati, Mei-Chun Hsu und Ming-Chien Shan. Bd. 2444. Lecture Notes in Computer Science. Berlin: Springer, 2002, S. 54–64. DOI: **10.1007/3-540-46121-3_8**.

[PY09] Mauro Pezzè und Michal Young. *Software testen und analysieren*. München: Oldenbourg, 2009.

[PZL08] Cesare Pautasso, Olaf Zimmermann und Frank Leymann. „Restful Web Services vs. "Big" Web Services: Making the Right Architectural Decision". In: *Proceedings of the 17th international conference on World Wide Web*. WWW '08. Beijing, China: ACM, 2008, S. 805–814. DOI: **10.1145/1367497.1367606**.

[Ra07] Leonard Richardson und Sam Ruby and. *Restful Web Services*. Sebastopol, CA, USA: O'Reilly Media, Mai 2007.

[RB12] John Resig und Bear Bibeault. *Secrets of the JavaScript Ninja*. Shelter Island, NY, USA: Manning Publications, 2012.

[RBG02] Nicholas Routledge, Linda Bird und Andrew Goodchild. „UML and XML schema". In: *Proceedings of the 13th Australasian database conference - Volume 5*. ADC '02. Melbourne, Victoria, Australia: Australian Computer Society, Inc., 2002, S. 157–166. DOI: **10.1145/563932.563924**.

[RBH07] J. Rech, C. Bogner und Volker Haas. „Using Wikis to Tackle Reuse in Software Projects“. In: *Software, IEEE* 24.6 (2007), S. 99–104. DOI: **10.1109/MS.2007.183**.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy und William Lorenson. *Object-Oriented Modeling and Design*. Upper Saddle River, NJ, USA: Prentice Hall, Inc., 1991.

[RCB10] Balasubramaniam Ramesh, Lan Cao und Richard Baskerville. „Agile requirements engineering practices and challenges: an empirical study“. In: *Information Systems Journal* 20.5 (2010), S. 449–480. DOI: **10.1111/j.1365-2575.2007.00259.x**.

[RCD+11] Jonathan Robie, Don Chamberlin, Michael Dyck, Daniela Florescu, Jim Melton und Jérôme Siméon. *XQuery Update Facility 1.0*. W3C Recommendation. W3C, März 2011.

[RCDS13a] Jonathan Robie, Don Chamberlin, Michael Dyck und John Snelson. *XML Path Language (XPath) 3.0*. W3C Candidate Recommendation. W3C, Jan. 2013. URL: **http://www.w3.org/TR/2013/CR-xpath-30-20130108/**.

[RCDS13b] Jonathan Robie, Don Chamberlin, Michael Dyck und John Snelson. *XQuery 3.0: An XML Query Language*. W3C Candidate Recommendation. W3C, Jan. 2013. URL: **http://www.w3.org/TR/2013/CR-xquery-30-20130108/**.

[Res08] P. Resnick. *Internet Message Format*. RFC 5322. Network Working Group, Okt. 2008. URL: **http://www.ietf.org/rfc/rfc5322.txt**.

[Ret12] Adam Retter. „RESTful XQuery“. In: *XML Prague 2012 – Conference Proceedings*. Hrsg. von Jirka Kosek. Prague, Czech Republic, 2012, S. 91–123. URL: **http://archive.xmlprague.cz/2012/files/xmlprague-2012-proceedings.pdf**.

[RL99] Fethi Rabhi und Guy Lapalme. *Algorithms: A Functional Programming Approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[RMB01] W.A. Ruh, F.X. Maginnis und W.J. Brown. *Enterprise application integration: a Wiley tech brief*. Wiley tech brief series. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[Roy70] Winston W. Royce. „Managing the Development of Large Software Systems“. In: *Proceedings of IEEE WESCON 26*. Los Angeles, CA, USA, Aug. 1970, S. 1–9.

[RPBP04] Kanda Runapongsa, Jignesh M. Patel, Rajesh Bordawekar und Sriram Padmanabhan. „XIST: An XML Index Selection Tool“. In: *Database and XML Technologies*. Hrsg. von Zohra Bellahsène, Tova Milo, Michael Rys, Dan Suciu und Rainer Unland. Bd. 3186. Lecture Notes in Computer Science. Berlin: Springer, 2004, S. 219–234. DOI: **10.1007/978-3-540-30081-6_16**.

[RTS13] K. Rajasri, S. Tamilarasi und S. Sathiyadevi. „A Security Based Multimedia Web Service Framework Supporting Multimedia Content Streaming". In: *International Journal of Advanced Research in Computer Science and Software Engineering* 3.2 (Feb. 2013), S. 157–168. URL: **http://www.ijarcsse.com/docs/papers/Volume_3/2_February2013/V3I2-0152.pdf**.

[Rus08] Craig Russell. „Bridging the Object-Relational Divide". In: *Queue* 6.3 (Mai 2008), S. 18–28. DOI: **10.1145/1394127.1394139**.

[San12] Javier Santamaría. „Time is of the Essence: Get Ready for SEPA. Act Now!" In: *European Payments Council Newsletter* 16 (Okt. 2012), S. 1–12. URL: **http://www.europeanpaymentscouncil.eu/pdf/EPC_Newsletter_291012_16.pdf**.

[Sav07] Alberto Savoia. „Beautiful Tests". In: Hrsg. von Andy Oram und Greg Wilson. Sebastopol, CA, USA: O'Reilly Media, 2007. Kap. Beautiful Tests, S. 85–103.

[SB08] Ken Schwaber und Mike Beedle. *Agile Software Development with Scrum*. Upper Saddle River, NJ, USA: Pearson Studium, 2008.

[SB13] Simon Stewart und David Burns. *WebDriver*. W3C Working Draft. W3C, März 2013. URL: **http://www.w3.org/TR/2013/WD-webdriver-20130312/**.

[SBH+07] Charles Safran, Meryl Bloomrosen, W. Edward Hammond, Steven Labkoff, Suzanne Markel-Fox, Paul C. Tang, Don E. Detmer und with input from the expert panel. „Toward a national framework for the secondary use of health data: an American Medical Informatics Association White Paper." In: *Journal of the American Medical Informatics Association* 14.1 (2007), S. 1–9. DOI: **10.1197/jamia.M2273**.

[SBS12] Harry M. Sneed, Manfred Baumgartner und Richard Seidl. *Der Systemtest: Von den Anforderungen zum Qualitätsnachweis*. München: Carl Hanser Verlag, 2012.

[Sch01] August-Wilhelm Scheer. *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen*. 4. Aufl. Berlin: Springer, 2001.

[Sch95] Ken Schwaber. „SCRUM Development Process". In: *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*. Hrsg. von Rebecca Wirfs-Brock. Austin, TX, USA, Okt. 1995, S. 117–134.

[SD00] J. Siedersleben und E. Denert. „Wie baut man Informationssysteme? Überlegungen zur Standardarchitektur". In: *Informatik-Spektrum* 23.4 (2000), S. 247–257. DOI: **10.1007/s002870000110**.

[SDH+09] Peter Suber, Cory Doctorow, Tim Hubbard, Peter Murray-Rust, Jo Walsh, Prodromos Tsiavos und et al. *Open Definition Version 1.1*. Techn. Ber. Open Knowledge Foundation, 2009. URL: **http://opendefinition.org/okd/deutsch/**.

[SFF+06] Thomas Söbbing, Axel Funk, Wolfgang Fritzemeyer, Holger Heinbuch, Sandra Neuhaus, Robert Niedermeier und Joachim Schrey. *Handbuch IT-Outsourcing: Recht, Strategie, Prozesse, IT, Steuern samt Business Process Outsourcing*. Heidelberg: C.F. Müller, 2006.

[SG71] Sidney L. Smith und Nancy C. Goodwin. „Alphabetic Data Entry via the Touch-Tone Pad: A Comment". In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 13.2 (1971), S. 189–190. DOI: **10.1177/001872087101300212**.

[SH05] P. Stahlknecht und U. Hasenkamp. *Einführung in die Wirtschaftsinformatik*. Springer-Lehrbuch. Berlin: Springer, 2005. URL: **http://stahlknecht-hasenkamp.de/**.

[SH06] T.C. Shan und W.W. Hua. „Taxonomy of Java Web Application Frameworks". In: *e-Business Engineering, 2006. ICEBE '06. IEEE International Conference on*. 2006, S. 378–385. DOI: **10.1109/ICEBE.2006.98**.

[Shn82] Ben Shneiderman. „The future of interactive systems and the emergence of direct manipulation". In: *Behaviour & Information Technology* 1.3 (1982), S. 237–256. DOI: **10.1080/01449298208914450**.

[Sil01] Wellington L. S. da Silva. *JSP and Tag Libraries for Web Development*. Thousand Oaks, CA, USA: New Riders Publishing, 2001.

[SKM00] Elmar J. Sinz, Bernd Knobloch und Stephan Mantel. „Web-Application-Server". In: *Wirtschaftsinformatik* 42.6 (2000), S. 550–552. DOI: **10.1007/BF03250773**.

[SLJ12] Weifeng Shan, Husheng Liao und Xueyuan Jin. „XML Concurrency Control Protocols: A Survey". In: *Web-Age Information Management*. Hrsg. von Zhifeng Bao, Yunjun Gao, Yu Gu, Longjiang Guo, Yingshu Li, Jiaheng Lu, Zujie Ren, Chaokun Wang und Xiao Zhang. Bd. 7419. Lecture Notes in Computer Science. Berlin: Springer, 2012, S. 299–308. DOI: **10.1007/978-3-642-33050-6_29**.

[SMD03] Arijit Sengupta, Sriram Mohan und Rahul Doshi. „XER — Extensible Entity Relationship Modeling". In: *Proceedings of the XML 2003 Conference*. Hrsg. von J. Harnad et al. Philadelphia, PA, USA, 2003, S. 8–12.

[Sof12] ISTAG – Information Society Technologies Advisory Group: Working Group on Software Technologies. *Software Technologies: The Missing Key Enabling Technology – Toward a Strategic Agenda for Software Technologies in Europe*. ISTAG Report (Draft). European Commission, Juli 2012. URL: **http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf**.

[Spä14] Rupert Späth. *Frühaufklärungssystem für Immobilienportfolios*. German. Wiesbaden: Springer Fachmedien, 2014. DOI: **10.1007/978-3-658-04248-6_4**. URL: **http://dx.doi.org/10.1007/978-3-658-04248-6_4**.

[Spi05] D. Spinellis. „Version control systems“. In: *Software, IEEE* 22.5 (2005), S. 108–109. DOI: **10.1109/MS.2005.140**.

[SS08] Martin Schumacher und Gabriele Schulgen-Kristiansen. *Methodik klinischer Studien: Methodische Grundlagen der Planung, Durchführung und Auswertung*. Heidelberg: Springer, 2008.

[SS77] John Miles Smith und Diane C. P. Smith. „Database abstractions: aggregation and generalization“. In: *ACM Trans. Database Syst.* 2.2 (Juni 1977), S. 105–133. DOI: **10.1145/320544.320546**.

[Sta07] Thorsten Stapelkamp. *Screen- und Interfacedesign: Gestaltung und Usability für Hard- und Software*. X.media.press Series. Heidelberg: Springer-Verlag GmbH, 2007.

[Stu07] Tony Stubblebine. *Regular Expression Pocket Reference*. Sebastopol, CA, USA: O'Reilly Media, Juli 2007.

[Suv12] Viraj Suvarna. „Investigator initiated trials (IITs)“. In: *Perspectives in Clinical Research* 3.4 (2012), S. 119–121. DOI: **10.4103/2229-3485.103591**.

[SV05] Thomas Stahl und Markus Völter. *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management*. Heidelberg: dpunkt.verlag, 2005.

[SV08] Shashi Shekhar und Ranga Raju Vatsavai. „The Handbook of Geographic Information Science“. In: Hrsg. von John Wilson und A. Stewart Fotheringham. Malden, MA, USA: Blackwell Publishing Ltd, 2008. Kap. Object-Oriented Database Management Systems, S. 111–143.

[SVOK11] Frank Schönthaler, Gottfried Vossen, Andreas Oberweis und Thomas Karle. *Geschäftsprozesse für Business Communities — Modellierungssprachen, Methoden, Werkzeuge*. München: Oldenbourg, 2011.

[SWI13a] S.W.I.F.T. SCRL. *Standards MT November 2014: Updated High-Level Information*. Techn. Ber. SWIFT, Nov. 2013. URL: **http://www.swift.com/assets/corporates/documents/our_solution/implementing_your_project_2011_standards_mx_general_information.pdf**.

[SWI13b] S.W.I.F.T. SCRL. *Standards MX Release November 2014*. Techn. Ber. SWIFT, Nov. 2013. URL: **http://www.swift.com/assets/corporates/documents/our_solution/implementing_your_project_2011_standards_mx_general_information.pdf**.

[SWW11a] Marco Skulschus, Marcus Wiederstein und Sarah Winterstone. *XML Schema*. Berlin: Comelio Medien, 2011.

[SWW11b] Marco Skulschus, Marcus Wiederstein und Sarah Winterstone. *XSLT, XPath und XQuery*. Berlin: Comelio Medien, 2011.

[SZ87] Karen E. Smith und Stanley B. Zdonik. „Intermedia: A case study of the differences between relational and object-oriented database systems“. In: *Conference proceedings on Object-oriented programming systems, languages and applications*. OOPSLA '87. Orlando, Florida, USA: ACM, 1987, S. 452–465. DOI: **10.1145/38765.38849**.

[TFO+10] Matthew D. Tipping, Victoria E. Forth, Kevin J. O'Leary, David M. Malkenson, David B. Magill, Kate Englert und Mark V. Williams. „Where did the day go?—A time-motion study of hospitalists“. In: *Journal of Hospital Medicine* 5.6 (2010), S. 323–328. DOI: **10.1002/jhm.790**.

[Thi11] Gunnar Thies. „Web-orientierte Architekturen: Eine Methode zur Konzeption, Planung, Umsetzung und Bewertung“. Dissertation. Westfälische Wilhelms-Universität Münster, 2011.

[Tid08] Doug Tidwell. *XSLT*. Sebastopol, CA, USA: O'Reilly, 2008.

[TLW03] Baltasar Trancón y Widemann, Markus Lepper und Jacob Wieland. „Automatic Construction of XML-Based Tools Seen as Meta-Programming“. In: *Automated Software Engineering* 10.1 (2003), S. 23–38. DOI: **10.1023/A:1021864801049**.

[Trz13] M. Trzaska. „WebODRA-A Web Framework for the Object-Oriented DBMS ODRA“. In: *WEB 2013, The First International Conference on Building and Exploring Web Based Environments*. 2013, S. 1–6.

[Ull12] Christian Ullenboom. *Java 7 – Mehr als eine Insel*. Bonn: Galileo Computing, 2012.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*. Bd. 2. New York, NY, USA: Computer Science Press, 1988.

[US07] Paul F. Uhlir und Peter Schröder. „Open Data for Global Science“. In: *Data Science Journal* 6 (Juni 2007), S. 36–53. DOI: **10.2481/dsj.6.OD36**.

[VH07] G. Vossen und S. Hagemann. *Unleashing Web 2.0: From Concepts to Creativity*. Burlington, MA, USA: Morgan Kaufmann, 2007.

[VHH12] Gottfried Vossen, Till Haselmann und Thomas Hoeren. *Cloud-Computing für Unternehmen – technische, wirtschaftliche, rechtliche und organisatorische Aspekte*. Heidelberg: dpunkt.verlag, 2012.

[Vli03] Eric van der Vlist. *XML Schema*. Köln: O'Reilly, 2003.

[Vli13a] Eric van der Vlist. „When MVC becomes a burden for XForms“. In: *XML London 2013 – Conference Proceedings* (Juni 2013), S. 15–34.

[Vos08] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. München: Oldenbourg Wissenschaftsverlag, 2008.

[Vos86] Gottfried Vossen. „Entwurf und Bearbeitung von Datenbanken im Universalrelationen-Datenmodell". Dissertation. Technische Hochschule Aachen, 1986.

[Wal09] Norman Walsh. *The DocBook Schema Version 5.0*. OASIS Standard. OASIS, 2009. URL: **http://docs.oasis-open.org/docbook/specs/docbook-5.0-spec-os.html**.

[Wal10] Dale Waldt. *Six strategies for extending XML schemas in a single namespace*. Techn. Ber. IBM Corporation, Jan. 2010. URL: **http://www.ibm.com/developerworks/library/x-xtendschema/**.

[Wal12] Priscilla Walmsley. *Definitive XML Schema*. Charles F. Goldfarb Definitive XML Series. Upper Saddle River, NJ, USA: Pearson Education, 2012.

[WBS13] Norman Walsh, Anders Berglund und John Snelson. *XQuery and XPath Data Model 3.0*. W3C Candidate Recommendation. W3C, Jan. 2013. URL: **http://www.w3.org/TR/2013/CR-xpath-datamodel-30-20130108/**.

[WCL+05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey und Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[Weg13] P. Wegrzynowicz. „Performance antipatterns of one to many association in hibernate". In: *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*. Kraków, Poland, Sep. 2013, S. 1475–1481.

[WES+13] Mario Winter, Mohsen Ekssir-Monfared, Harry M. Sneed, Richard Seidl und Lars Borner. *Der Integrationstest: Von Entwurf und Architektur zur Komponenten- und Systemintegration*. München: Carl Hanser Verlag, 2013.

[Wes12] Mathias Weske. *Business Process Management*. Berlin: Springer, 2012.

[WJR12] Matthias Wiesenauer, Christian Johner und Rainer Röhrig. „Secondary Use of Clinical Data in Healthcare Providers – an Overview on Research, Regulatory and Ethical Requirements". In: *Quality of Life through Quality of Information*. Hrsg. von J. Mantas, S.K. Andersen, B. Mazzoleni M.C. Blobel, S. Quaglini und A. Moen. Bd. 180. Studies in Health Technology and Informatics. IOS Press, Aug. 2012, S. 614–618. DOI: **10.3233/978-1-61499-101-4-614**.

[YJ08] Cong Yu und H. V. Jagadish. „XML schema refinement through redundancy detection and normalization". In: *The VLDB Journal* 17.2 (März 2008), S. 203–223. DOI: **10.1007/s00778-007-0063-0**.

[Zem12] Fred Zemke. „What's new in SQL:2011". In: *SIGMOD Rec.* 41.1 (Apr. 2012), S. 67–73. DOI: **10.1145/2206869.2206883**.

[ZKB06] Pieter van Zyl, Derrick G. Kourie und Andrew Boake. „Comparing the Performance of Object Databases and ORM Tools". In: *Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*. SAICSIT '06. Somerset West, South Africa: South African Institute for Computer Scientists und Information Technologists, 2006, S. 1–11. DOI: **10.1145/1216262.1216263**.

[ZKCB09] Pieter van Zyl, Derrick G. Kourie, Louis Coetzee und Andrew Boake. „The influence of optimisations on the performance of an object relational mapping tool". In: *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*. SAICSIT '09. Vanderbijlpark, Emfuleni, South Africa: ACM, 2009, S. 150–159. DOI: **10.1145/1632149.1632169**. URL: **http://doi.acm.org/10.1145/1632149.1632169**.

[Zlo75] Moshé M. Zloof. „Query by Example". In: *AFIPS National Computer Conference*. 1975, S. 431–438. URL: **http://doi.acm.org/10.1145/1499949.1500034**.

[ZR01] Weiping Zhang und Norbert Ritter. „The Real Benefits of Object-Relational DB-Technology for Object-Oriented Software Development". In: *Proceedings of the 18th British National Conference on Databases: Advances in Databases*. BNCOD 18. London, UK: Springer-Verlag, 2001, S. 89–104. URL: **http://dl.acm.org/citation.cfm?id=646103.681198**.

Web-Quellen

[1] *betterFORM Documentation*. URL: **http://betterform.wordpress.com/documentation/** (besucht am 19. 01. 2014).

[2] *Orbeon Forms Wiki Home*. URL: **http://wiki.orbeon.com/forms/** (besucht am 19. 01. 2014).

[3] *XForms 1.1 Implementation Status: betterFORM*. URL: **http://www.betterform.de/en/software.html** (besucht am 19. 01. 2014).

[4] *XForms 1.1 Test Suite (XSLTForms)*. URL: **http://www.agencexml.com/xforms-tests/testsuite/XForms1.1/Edition1/driverPages/html/** (besucht am 19. 01. 2014).

[5] *XForms - Orbeon Forms XForms 1.1 Implementation Report / Compliance Matrix (Work in Progress!)* URL: **http://wiki.orbeon.com/forms/doc/developer-guide/orbeon-forms-xforms-compliance-matrix** (besucht am 19. 01. 2014).

[6] *XQuery Test Suite Result Summary*. URL: **http://dev.w3.org/2006/xquery-test-suite/PublicPagesStagingArea/XQTSReport_XQTS_1_0_2.html** (besucht am 20. 01. 2014).

[7] *XSLTForms*. 2012. URL: **http://www.agencexml.com/xsltforms.htm** (besucht am 19. 01. 2014).

[8] *IBM Forms family*. 2013. URL: **http://www-03.ibm.com/software/products/en/ibmformfami** (besucht am 19. 01. 2014).

[9] Bert Appward. *Bert Appward's Field Guide to Web Applications | 2012 Edition*. 2012. URL: **http://www.html5rocks.com/webappfieldguide/** (besucht am 13. 08. 2013).

[10] Troels Arvin. *Comparison of different SQL implementations*. 2013. URL: **http://troels.arvin.dk/db/rdbms/** (besucht am 27. 08. 2013).

[11] Kent Beck, Mike Beedle, Arie van Bennekum u. a. *Manifest für Agile Softwareentwicklung*. 2001. URL: **http://agilemanifesto.org/iso/de/** (besucht am 27. 02. 2013).

[12] Mark Birbeck. *XForms Patterns: Incremental and 'Google Suggest'*. 2005. URL: **http://internet-apps.blogspot.de/2005/04/xforms-patterns-incremental-and-google.html** (besucht am 01. 11. 2013).

[13] Guy Burstein. *How To: Model Inheritance in Databases*. 2007. URL: **http://blogs.microsoft.co.il/blogs/bursteg/archive/2007/09/30/how-to-model-inheritance-in-databases.aspx** (besucht am 06. 10. 2013).

[14] CenterWatch: Bio-IT World. *EDC Adoption in Clinical Trials: A 2008 Analysis*. 2008. URL: **http://www.bio-itworld.com/EDC-Adoption.aspx** (besucht am 12. 09. 2013).

[15] Eco. *ODM Tutorial 6: StudyEventDefs, Forms and ItemGroupRefs | eClinicalOpinion*. 2009. URL: **http://eclinicalopinion.blogspot.de/2009/05/odm-tutorial-6-studyeventdefs-forms-and.html** (besucht am 13. 09. 2013).

[16] eXist-db Project. *eXist-db Documentation*. 2013. URL: **http://exist-db.org/exist/apps/doc/** (besucht am 19. 01. 2014).

[17] eXist-db Project. *eXist-DB Documentation: XQuery Update Extension*. 2013. URL: **http://exist-db.org/exist/apps/doc/update_ext.xml** (besucht am 15. 04. 2013).

[18] Neal Ford. *Functional thinking: Functional design patterns, Part 1 – How patterns manifest in the functional world*. März 2012. URL: **https://www.ibm.com/developerworks/java/library/j-ft10/index.html** (besucht am 23. 01. 2014).

[19] Renate Gömpel. *Deutsche Nationalbibliothek – Formate und Schnittstellen*. 2013. URL: **http://www.dnb.de/formate**.

[20] Jan Goyvaert. *How to Find or Validate an Email Address*. 2010. URL: **http://www.regular-expressions.info/email.html** (besucht am 05. 04. 2013).

[21] Institute for System Programming RAS. *Sedna Documentation*. 2012. URL: **http://sedna.org/documentation.html** (besucht am 19. 01. 2014).

[22] Library of Congress' Network Development and MARC Standards Office. *MARC 21 XML Schema*. Apr. 2012. URL: **http://www.loc.gov/standards/marcxml/** (besucht am 03. 02. 2014).

[23] Liquid Technologies Limited. *XML Standards Library - Documentation and Diagrams for Open XML Standards*. 2012. URL: **http://schemas.liquid-technologies.com/** (besucht am 23. 01. 2014).

[24] MarkLogic Corporation. *MarkLogic 7 Product Documentation*. 2014. URL: **http://docs.marklogic.com/** (besucht am 10 .02. 2014).

[25] Joe McKendrick. *IBM acknowledges bypassing UDDI; calls for new SOA registry standard*. 2007. URL: **http://www.zdnet.com/blog/service-oriented/ibm-acknowledges-bypassing-uddi-calls-for-new-soa-registry-standard/864** (besucht am 23. 01. 2014).

[26] Darin McBeath und Curt Kohler. *xqDoc*. 2011. URL: **http://xqdoc.org/index.html** (besucht am 30. 09. 2013).

[27] Ted Neward. *The Vietnam of Computer Science*. 2006. URL: **http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx** (besucht am 17. 03. 2013).

[28] Oracle Corporation. *Java EE Reference*. URL: **http://www.oracle.com/technetwork/java/javaee/documentation/index.html** (besucht am 17. 06. 2013).

[29] Tom Preston-Werner. *Semantic Versioning 2.0.0*. 2013. URL: **http://semver.org/spec/v2.0.0.html** (besucht am 18. 10. 2013).

[30] ProgrammableWeb.com. *API Directory*. 2013. URL: **http://www.programmableweb.com/apis/directory** (besucht am 17. 08. 2013).

[31] software AG. *tamino XML Server Version 8.2.2*. 2013. URL: **http://documentation.softwareag.com/webmethods/tamino/ins82/general/print.htm** (besucht am 19. 01. 2014).

[32] Nancy J Stark. *Registry Studies: Why and How*. 2011. URL: **http://clinicaldevice.typepad.com/cdg_whitepapers/2011/07/registry-studies-why-and-how.html**.

[33] Joern Turner. *XForms patterns – the submission chain*. März 2011. URL: **https://betterform.wordpress.com/2011/03/17/xforms-patterns-the-submission-chain/** (besucht am 01. 11. 2013).

[34] Eric van der Vlist. *XForms Unit*. 2013. URL: **http://eric.van-der-vlist.com/blog/2013/08/20/xforms-unit/** (besucht am 07. 11. 2013).

[35] V-Modell®XT Autoren. *V-Modell XT*. 2012. URL: **http://www.cio.bund.de/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.htm** (besucht am 27. 02. 2013).

[36] W3C. *Web Services @ W3C*. URL: **http://www.w3.org/2002/ws/** (besucht am 08. 08. 2013).

[37] W3C. *Schema - W3C*. 2010. URL: **http://www.w3.org/standards/xml/schema** (besucht am 02. 04. 2013).

[38] Wikibook Contributors. *XForms – Tutorial and Cookbook*. URL: **https://en.wikibooks.org/w/index.php?title=XForms&oldid=2559832** (besucht am 01. 11. 2013).

[39] XML Query Working Group. *XML Query Test Suite*. 2010. URL: **http://dev.w3.org/2006/xquery-test-suite/** (besucht am 11. 11. 2013).

Abkürzungsverzeichnis

1NF	erste Normalform
2NF	zweite Normalform
3NF	dritte Normalform
ABAC	Attribute Based Access Control
AOM	Asset Oriented Modelling
API	Application Programming Interface
ARIS	Architektur integrierter Informationssysteme
BCNF	Boyce-Codd-Normalform
BPMN	Business Process Markup Language
BSD	Berkeley Software Distribution
CDA	Clinical Document Architecture
CDISC	Clinical Data Interchange Standards Consortium
CI	Continuous Integration
CRF	Case Report Form
CSS	Cascading Stylesheets
CSV	Comma-separated Values
DBMS	Datenbankmanagementsystem
DRY	Don't Repeat Yourself
DTD	Document Type Definition
DWH	Data-Warehouse
EBNF	Erweiterte Backus-Naur-Form
EDC	Electronic Data Capture
EHR	Electronic Health Record
EPK	Ereignisgesteuerte Prozesskette
ERM	Entity-Relationship-Modell
ETL	Extrahieren, Transformation, Laden
FIXML	Financial Information eXchange Markup Language
FLWOR	FOR, LET, WHERE, ORDER BY, RETURN
FpML	Financial products Markup Language
GOM	Grundsätze ordnungsmäßiger Modellierung
GPL	GNU General Public License
GUI	Graphical User Interface
GXSL	Graphical XML Schema Definition Language
HL7	Health Level 7
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IDE	Entwicklungsumgebung
ISO	International Organization for Standardization
Java EE	Java Enterprise Edition
JAXB	Java Architecture for XML Binding
JAX-WS	Java API for XML Web Services
JSF	JavaServer Faces
JSP	JavaServer Pages
KIS	Krankenhausinformationssystem
LGPL	GNU Lesser General Public License
LINQ	Language Integrated Query
MT	Message Type
MTOM	Message Transmission Optimization Mechanism
MVC	Model View Controller
MX	Standards MX messages
OASIS	Organization for the Advancement of Structured Information Standards
ODM	Operational Data Model
OKF	Open Knowledge Foundation
OSCRE	Open Standards Consortium for Real Estate
PDF	Portable Document Format
REST	Representational State Transfer
SaaS	Software-as-a-Service
SEPA	Single Euro Payments Area
SLA	Service Level Agreement
SOA	Serviceorientierte Architektur
SQL	Structured Query Language
SSO	Single Sign-on

SVN	Apache Subversion
SWIFT	Society for Worldwide Interbank Financial Telecommunication
SWOT	Strengths, Weaknesses, Opportunities, Threats
UDDI	Universal Description, Discovery and Integration
UKM	Universitätsklinikum Münster
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WOA	Web-orientierte Architektur
WSDL	Web Services Description Language
x4T	exchange for trials
XDM	XQuery and XPath Data Model
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XOP	XML-binary Optimized Packaging
XQTS	XML Query Test Suite
XSD	XML Schema Definition
XSL FO	XSL Formatting Objects
XSLT	Extensible Stylesheet Language Transformations

Webanwendungsentwicklung mit XML-Techniken

Christian Forster

Webanwendungen machen einen großen Teil der gegenwärtig entwickelten betrieblichen Software aus. Sie werden oft nach dem Schichtenmodell entwickelt, bei dem die Funktionen der Software häufig in Datenhaltung, Geschäftslogik und Präsentation aufgeteilt entworfen und innerhalb der einzelnen Schichten mit der isoliert betrachtet jeweils vorteilhaftesten Technik implementiert werden. Beim Datentransfer zwischen den Schichten entstehen durch inkompatible Datenmodelle Brüche, die durch zusätzliche Technik überbrückt werden müssen.

Mit den in jüngerer Zeit entstandenen Spezifikationen der XML-Technikfamilie und deren Implementierungen stehen Techniken zur Verfügung, die sich in allen Schichten einer Webanwendung einsetzen lassen und durchgängig auf einem XML-Datenmodell basieren. Die Arbeit beschreibt einen Softwareentwurf, der beginnend mit XML-Netzen in Analyse- und Entwurfsphase eine Webanwendung implementiert, die XML-Datenbanken zur Datenhaltung verwendet, Geschäftslogik in XQuery realisiert und XForms zur Präsentation verwendet. Die Architektur wird anhand eines Anwendungsbeispiels entwickelt. Es wird eine domänenübergreifende Abstraktion der Softwarearchitektur vorgestellt und Hinweise auf weitere Anwendungsmöglichkeiten werden gegeben.

Mittels der XML-basierten Architektur lassen sich Webanwendungen, die strukturierte Daten verarbeiten, ohne Brüche zwischen den Schichten entwickeln. Die bewährte Aufteilung in Schichten wird beibehalten, jedoch ermöglicht die konsequente Verwendung von XML-Techniken einen Verzicht auf Datenmapping, vermeidet insbesondere objektrelationale Transformationen und ermöglicht somit die Fokussierung auf die Entwicklung von (Geschäfts-)Nutzen stiftender Funktionalität.