Konzeption und Entwicklung eines Generators ereignisbasierter Patientendaten für ein virtuelles Gesundheitssystem

Diplomarbeit

Sergei Wagner, 161240

Referent: **Prof. Dr. Christian Fegeler**

Korreferent: **Prof. Dr. Martin Haag**

21. März 2011

Danksagung

Ich möchte mich an dieser Stelle bei all denen bedanken, die mich bei der Erstellung meiner Diplomarbeit unterstützt haben.

Ganz besonders bedanken möchte ich mich bei meinem Referenten, Herrn Prof. Dr. C. Fegeler für die Betreuung und Unterstützung während dieser Diplomarbeit.

Ebenfalls danken möchte ich meinem Korreferenten Herrn Prof. Dr. M. Haag für die Unterstützung.

Großer Dank gilt meiner Frau, die mich während des Studiums und vor allem bei der Diplomarbeit moralisch unterstützt und motiviert hat. Ebenfalls danken möchte ich besonders meinen Eltern, die mir das Studium ermöglicht und mich in guten sowie auch in schlechten Zeiten unterstützt haben.

Abschließend danke ich Simon Pezold und Sabrina Lang für das Korrekturlesen meiner Diplomarbeit.

Abstract

Das deutsche Gesundheitswesen ist ein dichtes Netzwerk bestehend aus einer Vielzahl von unterschiedlichen Akteuren im komplexen Zusammenspiel. Ständige Gesundheitsreformen aufgrund steigender Ausgaben im Gesundheitswesen sowie Fortschritte in der Medizin nehmen Einfluss auf die Informationsverarbeitung in diesem Netzwerk. Das hat zur Folge, dass immer mehr Anwendungssysteme zum Einsatz kommen, die hinsichtlich ihrer Zusammenarbeit besondere Herausforderungen stellen. Häufig können die komplexen Abläufe bei der Zusammenarbeit der Anwendungssysteme erst durch ein Modell, welches die Realität abstrahiert, verstanden werden. In diesem Zusammenhang wird das deutsche Gesundheitswesen durch ein virtuelles Gesundheitssystem modelliert, welches die Akteure des deutschen Gesundheitswesens nachbildet. Im Rahmen der vorliegenden Diplomarbeit wird die Population durch einen Generator abgebildet. Der Generator generiert anhand der Patientendaten der Population ereignisbasierte Nachrichtenprofile, die dem virtuellen Gesundheitssystem zur Weiterverarbeitung zur Verfügung gestellt werden.

Im Rahmen der Diplomarbeit wird das Spiralmodell eingesetzt, welches zum Einen hinsichtlich der definierten Projektphasen iterativ und inkrementell vorgeht und zum Anderen die im Projekt mögliche Risiken berücksichtigt. Die Entwicklung des Generators erfolgt in Java.

Der Generator stellt ein breites Spektrum unterschiedlicher Funktionalitäten zur Verfügung. Zu den Hauptmerkmalen gehören folgende Aspekte:

- Import der Patientendaten
- Parametrierung der Nachrichtenprofile über ein Webinterface
- Simulation der Nachrichtenprofile nach der Zeit

Das Generator-Projekt konnte im Laufe der vorliegenden Diplomarbeit aufgrund seiner Komplexität und seines Umfangs nicht vollständig realisiert werden. Das Konzept deckt die Anforderungen des Generators vollständig ab. Hinsichtlich der Implementierung liegen Ansätze vor, die im Fall einer Projektfortführung zugrunde gelegt werden können. Zum aktuellen Zeitpunkt liegt kein vergleichbares Projekt vor, somit definiert das Generator-Projekt den ersten Meilenstein in der Entwicklung des virtuellen Gesundheitssystems.

Inhaltsverzeichnis

1. Einleitung

Das deutsche Gesundheitswesen ist ein dichtes Netzwerk bestehend aus einer Vielzahl von unterschiedlichen Akteuren im komplexen Zusammenspiel. Als Teil des Sozialversicherungssystems wird es seit 1883 durch wesentliche Meilensteine geprägt. 1883 wird das Krankenversicherungsgesetz verabschiedet. Trotz der Strukturveränderungen in den 1970er Jahren, dem GKV-Modernisierungsgesetz (Gesetzliche Krankenversicherung) 2000/2001 und der Einführung des DRG-Systems (Diagnosis Related Groups) 2003/2004 bleibt das deutsche Gesundheitswesen in seiner Grundstruktur weitgehend unverändert (Franke 2008 S.190). In der Abbildung 1 werden grob die Beziehungen der Akteure im deutschen Gesundheitswesen dargestellt.

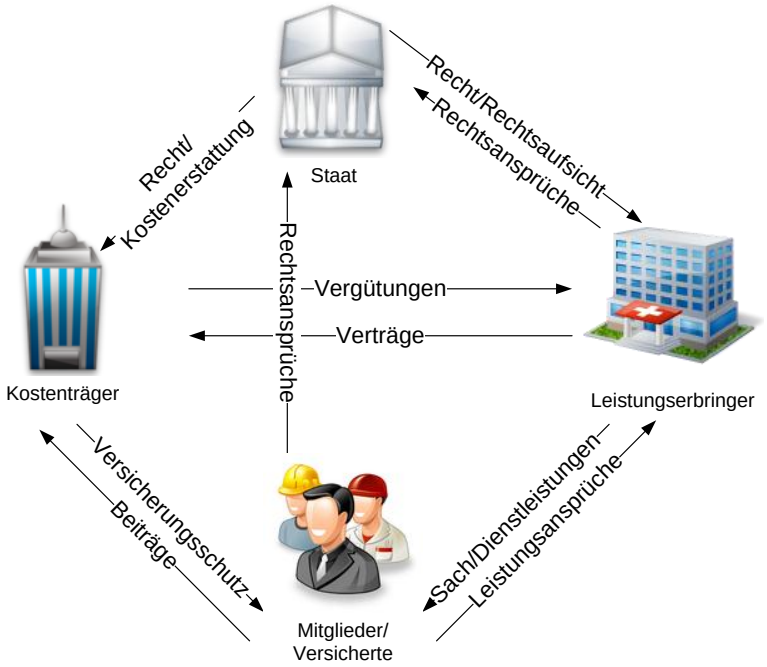


Abbildung 1 Beziehungen der Akteure im Gesundheitswesen (Franke 2008 S.190)

Aus gesundheitsökonomischer Sicht befindet sich das deutsche Gesundheitssystem heutzutage in einem Dilemma: wachsende Kostenproblematik bei steigender Nachfrage der Gesundheitsleistungen (Schlegel 2010 S.73). Hierfür sind im Wesentlichen die folgenden Ursachen und Zusammenhänge verantwortlich: Fortschritte in der Technologie und Medizin sorgen für mehr Angebot an Gesundheitsleistungen. Die Bevölkerung wird zunehmend älter und chronisch

kränker. Steigende Ausgaben (Statistisches Jahrbuch S.260) im Gesundheitswesen veranlassen Strukturveränderungen und Reformen zur Kostendämpfung, die bereits seit den 1970ern bis dato das deutsche Gesundheitswesen begleiten.

Bei der Organisation und Erbringung der Dienstleistungen im Gesundheitswesen entsteht ebenso ein komplexer wie dichter Informationsfluss. Seit den Anfängen des Fachs Medizinische Informatik in den 1970er Jahren hat sich in der Verarbeitung von Daten im Gesundheitswesen viel getan. Nahezu jede Institution im Gesundheitswesen verfügt heute über besondere Anwendungssoftwaresysteme, die entweder einfache Funktionen oder komplexe Prozesse abbilden. Dabei können an einem Prozess mehrere Anwendungssoftwaresysteme beteiligt sein, die zum Einen physisch voneinander entfernt (verteilt) und zum Anderen systemspezifisch verschieden (heterogen) sein können. Medizinisch relevante Daten entstehen bereits innerhalb einer Versorgungseinrichtung, wie beispielsweise einem Krankenhaus, in aller Regel in verteilten und heterogenen Anwendungssoftwaresystemen (Ingenerf 2004 S.1). Probleme mit der Interoperabilität, also der Zusammenarbeit der Komponenten, stellen dabei eine besondere Herausforderung dar. Standards wie HL7 (Health Level 7) oder DICOM (Digital Imaging and Communications in Medicine) werden geschaffen – Organisationen wie IHE (Integrating the Healthcare Enterprise) werden gegründet um der Problematik bei der Interaktion heterogener Anwendungssysteme entgegenzuwirken. Das wesentliche Ziel ist, zum Einen die Qualität der Patientenversorgung zu steigern und zum Anderen mehr Effizienz zu schaffen um im bereits erwähnten Dilemma steigende Kosten einzusparen.

Wie schon angeführt wird das deutsche Gesundheitswesen durch eine Vielzahl von Akteuren, die in einem komplexen Geflecht von Beziehungen zueinander stehen, geprägt. Komplexe Vorgänge werden üblicherweise durch Modelle nachgebildet. Die Organisation IHE versucht beispielsweise die medizinischen Prozessabläufe aus der Praxis als Anwendungsfälle abzuleiten und daraus relevante Standards zu identifizieren sowie technische Leitfäden zu erstellen (Haas 2006 S.306). Diese können dann von einem Hersteller in seinem Produkt umgesetzt und getestet werden. Die Alternative wäre ein Gesundheitssystem, das nur für Testzwecke zur Verfügung steht, in dem sämtliche Akteure, Produkte der Hersteller und Standards virtuell miteinander interagieren können, um daraus den Nutzen, die Kosten sowie die Fehlerquellen zu ermitteln. Jedoch ist es allein aus Kostengründen undenkbar. Dennoch ist es im Rahmen des Möglichen ein Modell einzuführen, welches das deutsche Gesundheitswesen abstrahiert.

Die Population, in dem Fall die Versicherten der Krankenkasse, werden in der vorliegenden Diplomarbeit durch den Generator abstrahiert. Der Generator ist somit einst der Akteure im virtuellen Gesundheitssystem. Der Anwendungszweck des Generators besteht darin, die Komplexität der Informationsflüsse im Gesundheitswesen, die beispielsweise durch Population entstehen können, hinsichtlich der Interoperabilität nachzubilden. Im nächsten Kapitel wird die Beziehung des Generators zum virtuellen Gesundheitssystem konkret vorgestellt.

1.1. Zielsetzung

Das Ziel der vorliegenden Arbeit ist die Konzeption und Entwicklung eines Generators ereignisbasierter Patientendaten für ein virtuelles Gesundheitssystem. Das virtuelle Gesundheitssystem bildet das sozialversicherungsrechtliche Dreieck des deutschen Gesundheitssystems nach. Das Dreieck beschreibt die Beziehung zwischen Kostenträger, Leistungserbringer und einer Population. Im Kapitel 2.1 wird das virtuelle Gesundheitssystem näher beschrieben. Der Generator bildet den Eckpunkt „Population“ aus Falldaten nach §21 KHEntgG (Krankenhausentgeltgesetz) ab. Die Falldaten nach §21 KHEntgG werden zur Vereinfachung als §21-Daten bezeichnet. Basierend auf dem Dateninput und den Steuerungsparametern für den Simulationslauf werden als Output Dateien in einem XML-Format (Extensible Markup Language) generiert, die über einen Kommunikationsserver im virtuellen Gesundheitssystem weiterverarbeitet werden. Der Generator soll als Webservice konzipiert werden. Über das Benutzerinterface steuert der Benutzer folgende vier wesentliche Aspekte für eine Simulation:

1. Profil der zugrunde gelegten Population
2. Ereignisse
3. Ereignisbasierte Nachrichtenprofile
4. Faktor der Zeitraffung

Diese Einstellungen ermöglichen dem Benutzer bestimmte Testszenarios zu simulieren. Der Generator an sich soll nach dem Start eines Simulationslaufes automatisiert ablaufen und ein Protokoll erstellen. Als Testszenario innerhalb der Diplomarbeit wird die Generierung von Datensätzen für eine Simulation einer

Kommunikation nach §301 SGB V (Sozialgesetz fünftes Buch) gewählt. Die Kommunikation nach §301 SGB V wird in Kapitel 2.2 näher erläutert.

1.2. Vorgehensweise

In der Regel lässt sich die Komplexität im System oder in einem Projekt durch Zerlegung in kleinere Einheiten reduzieren. Im Rahmen dieser Arbeit wird der Projektgegenstand, die Konzeption und Entwicklung eines Generators zur Simulation ereignisbasierter Nachrichten, in Gestaltungsbereiche zerlegt, die durch folgende Fragen charakterisiert werden.

1. Wer ist vom Projekt betroffen oder daran beteiligt?
2. Welche Themen sind für das Projekt relevant?
3. Welche Beziehungen werden durch Gestaltungsbereiche aufgebaut?
4. Welche Faktoren haben Einfluss auf das Projekt?

Die Beantwortung der Fragen ergibt Gestaltungsbereiche, die sowohl das System als auch das Projekt betreffen. Die Gestaltungsbereiche des Generators beziehen sich intern auf seine Komponenten und nach außen auf sein Umfeld. In Kapitel 2 wird die Abgrenzung des Generators nach außen und nach innen näher erläutert. Projektspezifische Gestaltungsbereiche beziehen sich üblicherweise auf ein phasenorientiertes Vorgehen, das in der Softwareentwicklung verwendet wird. Die Konzeption und Entwicklung des Generators lehnt sich in diesem Projekt an das Spiralmodell an, das in Kapitel 4 näher vorgestellt wird.

1.3. Aufbau der Diplomarbeit

In Kapitel 1 wird zunächst auf die aktuelle Problematik im Gesundheitswesen eingegangen. Mit der Zielsetzung wird die Aufgabenstellung der vorliegenden Diplomarbeit beschrieben. In Kapitel 2 werden Überlegungen für die Konzeption und Umsetzung des Generators angeschnitten. Das deutsche Gesundheitswesen wird auf das virtuelle Gesundheitssystem abstrahiert. Die Beziehungen der Akteure werden

zunächst aus der datenverarbeitenden Sicht beschrieben. Das Wissen technischer Natur, das primär mit der Umsetzung des Generators zu tun hat, wird dabei auf das Kapitel 3 ausgelagert. In Kapitel 3 werden mögliche Technologien, zur Umsetzung des Generators als eine Web-Anwendung, in Betracht gezogen. Das Kapitel 4 beschreibt den Generator aus der Projektperspektive. Das Generator-Projekt wird durch Iterationen und dessen Meilensteine beschrieben, dabei wird das Spiralmodell nach Boehm als Vorgehensmodell verwendet. Anschließend wird der Generator auf Alternativen von der architektonischen Seite, sowie auf mögliche Risiken, untersucht. In Kapitel 5 werden die Anforderungen an den Generator, sowie deren Verhalten, anhand der UML-Diagramme (Unified Modeling Language) analysiert. Desweiteren werden Anforderungen an die §21-Daten ermittelt und im Bezug auf das Risikomanagement wird ein Prototyp definiert. In Kapitel 6 wird das Domänenmodell des Generators als Klassendiagramm vorgestellt. Ebenfalls werden ansatzweise die Beziehungen darüber liegender Systemschichten als UML-Diagramm dargestellt. Das Kapitel 7 beschäftigt sich mit der Realisierung des Prototyps. Desweiteren werden dort Tests spezifiziert. Abschließend werden in Kapitel 8 die Ergebnisse präsentiert und in Kapitel 9 die Vorgehensweise im Projekt diskutiert sowie mögliche Erweiterungen vorgestellt. Zu den externen Dokumenten gehören das Lastenheft und das Pflichtenheft sowie Dokumente bezüglich der §21-Daten.

2. Vorbetrachtungen zur Realisierung

Zur Realisierung des Generator-Projektes müssen neben der technologischen auch die rechtlichen Aspekte herangezogen werden, die das virtuelle Gesundheitssystem letztendlich beeinflussen. In diesem Kapitel wird zunächst das Dreieck aus Kapitel 1.1 aufgegriffen und auf rechtliche Aspekte untersucht.

Im Hinblick auf Kosteneinsparung im Gesundheitswesen werden mit dem Gesundheitsreformgesetz im Jahr 2000 Voraussetzungen für das im Jahr 2004 eingeführte G-DRG-System (G-DRG German Diagnosis Related Groups) geschaffen. Zur Vereinfachung wird der Begriff DRG verwendet. Die DRGs definieren die Leistungen, die anhand der Haupt- und Nebendiagnose der Patienten für den einzelnen Fall in Fallgruppen klassifiziert werden. Die festgestellten DRGs werden zwischen Leistungsträger und Kostenträger als Abrechnungsgrundlage gemeldet. Mit dem DRG-System wird eine umfassende Änderung der Vergütung voll- und teilstationärer Leistungen im Krankenhausbereich eingeleitet (Johner & Haas 2009 S.124). Die Aufgaben im Zusammenhang mit der Einführung, Pflege und Weiterentwicklung des neuen Vergütungssystems werden ab 2001 der *InEK* (Institut für das Entgeltsystem im Krankenhaus) übertragen. Vertragsparteien der *InEK* sind die Spitzenverbände der Krankenkassen, der Verband der privaten Krankenversicherung sowie die Krankenhausgesellschaft. Gemäß § 21 des KHEntgG müssen die dem Anwendungsbereich des KHEntgG unterliegenden Krankenhäuser ihre Leistungsdaten jeweils zum 31. März für das jeweils vorangegangene Kalenderjahr an die Datenstelle übermitteln. Die Übermittlung der Daten wird durch entsprechende Dokumente, zur Vereinfachung als Vereinbarung benannt, spezifiziert, die in Kapitel 2.2 näher erläutert werden. Zur Annahme von Daten wird von den Vertragsparteien nach § 17b Abs. 2 Satz 1 KHG (Krankenhausfinanzierungsgesetz) die Gesellschaft *3M Medica* als Datenstelle benannt (Datenstelle *InEK* GmbH) Die Gründung der *InEK* ist ein Beispiel für das Selbstverwaltungsprinzip im deutschen Gesundheitswesen. Der Staat tritt als Gesetzgeber auf und gibt den rechtlichen Rahmen vor. Die Verwaltungsaufgaben werden dabei an rechtlich verselbstständigte Organisationen delegiert. Dazu gehören beispielsweise gesetzliche Krankenkassen, die als Kostenträger fungieren oder Krankenhäuser, die zu den Leistungserbringern gehören.

2.1. Virtuelles Gesundheitssystem

Der Generator steht mit dem Leistungserbringer und dem Kostenträger über einen Kommunikationssever in Verbindung. Gemeinsam bilden sie das virtuelle Gesundheitssystem aus Kapitel 1.1 nach. Die Rolle des Staates wird durch Gesundheitsreformen vertreten. Die Umsetzung des §21 des KHEntgG bewirkt die Gründung von *InEK*. Eine weitere Gesundheitsreform kann dazu führen, dass ein weiterer Akteur im Gesundheitswesen die Rolle des Staates vertritt. Als Beispiel wird am Rande ein weiteres Institut erwähnt, das sich der Bewertung der Qualität und Wirtschaftlichkeit im Gesundheitswesen widmet – *IQWiG* (Institut für Qualität und Wirtschaftlichkeit im Gesundheitswesen). Der §35b – Bewertung des Nutzens und der Kosten von Arzneimitteln – des SGBV wird unter anderem durch *IQWIG* umgesetzt (*IQWiG* - Gesetzliche Grundlage). Nach dem §21KHEntg-Gesetz werden Leistungsdaten vom Leistungserbringer an *3M Medica* zugestellt. *3M Medica* gibt die Daten nur an die *InEK* weiter. Insofern muss der Generator seine Daten in irgendeiner Form von einer anderen Instanz bekommen. Im Zusammenhang mit der Vereinbarung der *InEK* und den Leistungsdaten eines Krankenhauses ist es möglich ein Dateninput nach §21KHEntgG für den Generator zu definieren. Die Vereinbarung der *InEK* und die anonymisierten Falldaten nach §21KHEntgG bilden die Grundlage für das Dateninput des Generators. Ein weiterer wichtiger Akteur im virtuellen Gesundheitssystem ist der Kommunikationsserver. Er versorgt das virtuelle Gesundheitssystem mit Nachrichten, die aufgrund der §21-Daten vom Generator generiert werden. In der Regel ist ein Kommunikationsserver ein Bestandteil eines Krankenhausinformationssystems (KIS). Allerdings erfolgt die Kommunikation zwischen Leistungserbringer und *3M Medica* nicht über einen Kommunikationsserver. In Kapitel 2.2 wird der Datenaustausch zwischen *3M Medica* und Leistungserbringer sowie Leistungserbringer und Kostenträger gesondert beschrieben. Es ist nicht auszuschließen, dass aufgrund neuer Gesundheitsreformen weitere Akteure im Gesundheitswesen definiert werden, jedoch sind sie im Zusammenhang dieser Arbeit zunächst nicht von Bedeutung.

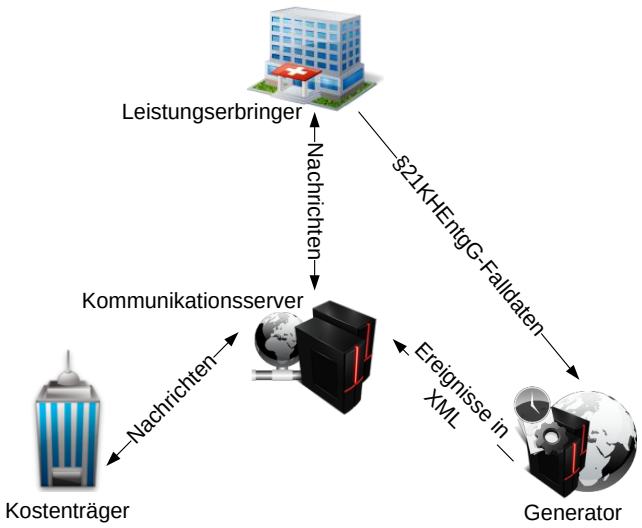


Abbildung 2 Virtuelles Gesundheitssystem als Dreieck

Das konkrete Dreieck aus Kapitel 1.1 wird in der Abbildung 2 wie folgt dargestellt. Der Generator versorgt den Kommunikationsserver mit Ereignissen in XML-Dateien. Je nach Ereignistyp werden durch den Kommunikationsserver Nachrichten entweder an ein Krankenhaus oder eine Krankenkasse zugestellt. Der Generator selbst bekommt seine Daten vom Leistungserbringer. Leistungserbringer und Kostenträger können auf einfache Weise durch Ordner repräsentiert werden, die der Kommunikationsserver im dateibasierten Fall per File-Transfer-Protocol (FTP) ansprechen kann. Es sind auch weitere Protokolle möglich, wie beispielsweise das Hypertext-Transfer-Protocol (HTTP). Im Hinblick auf Erweiterungen des Generators können durchaus Kommunikationen in Richtung des Generators erfolgen (in der Abbildung 2 nicht dargestellt), um beispielsweise so durch Krankenkasse oder Krankenhaus geänderte Nachrichten im Generator wieder einlesen zu können und somit Manipulationen an Daten aufdecken zu können. Dieser Aspekt wird bei der Prüfung auf Korrektheit der DRG-Daten relevant.

2.2. Kommunikation

Die Datenübermittlung im deutschen Gesundheitswesen wird durch rechtliche Aspekte geregelt. Der §21 des KHEntgG wurde bereits im vorigen Kapitel vorgestellt er regelt die Datenübermittlung zwischen einem Krankenhaus und der Datenstelle *3M*

Medica. Die Datenübermittlung wird durch Dokumente beschrieben, die von Vertragsparteien der *InEK* festgelegt worden sind. Auf der Homepage der *InEK* unter *Dokumente der Datenlieferung* (Datenlieferung gem. §21 KHEntgG, InEK GmbH) werden die Dokumente gelistet. Im Zusammenhang der Arbeit sind für den Generator folgende Dokumente relevant: *Datensatzbeschreibung*, *Fehlerverfahren* und das *Merkblatt*. Sie definieren die Quelle und Senke bei der Übermittlung von Daten sowie das für die Übermittlung benötigte Datenformat und Maßnahmen in einem Fehlerfall. Im Kapitel 2.3 wird darauf gesondert eingegangen.

Die Datenübermittlung zwischen einem Kostenträger und einem Leistungserbringer wird durch einen weiteren rechtlichen Aspekt geregelt. Die nach §108 SGBV zugelassenen Krankenhäuser sind verpflichtet, den Krankenkassen bestimmte Angaben bei einer Krankenhausbehandlung elektronisch oder auf einem Datenträger zu übermitteln. Die Angaben und das Verfahren zur Datenübermittlung werden durch den §301 des SGBV festgelegt und können aus voriger Quelle entnommen werden. Dieses Verfahren zur Datenübermittlung zwischen Krankenkasse und Krankenhaus kann beispielsweise für die Testsimulation des Generators herangezogen werden. Das Verfahren der Datenübermittlung wird durch das EDIFACT-Datenaustauschformat (Electronic Data Interchange For Administration, Commerce and Transport) repräsentiert. In (Bärwolff u. a. 2006 S.163) wird der Datenaustausch im Zusammenhang mit dem EDIFACT-Format näher beschrieben.

2.3. Datenumfeld

In einer Versorgungseinrichtung wie beispielsweise dem Krankenhaus entstehen bei einer Patientenbehandlung Daten, die zwecks der Informations- und Wissenslogistik mit Hilfe von Informationssystemen weiterverarbeitet werden. Wenn diese Daten an eine Krankenkasse oder Datenstelle übermittelt werden, müssen sie denselben Inhalt aufweisen, wie vor der Übermittlung. Dazu bedarf es Syntax- und Semantikregeln, durch die das Datenformat bestimmt wird. Im vorigen Kapitel wurden Dokumente der Datenstelle *InEK* vorgestellt, die die notwendigen Vereinbarungen treffen um eine fehlerfreie Datenübermittlung zwischen Krankenhaus und *InEK* zu gewährleisten. Diese Dokumente bilden die Grundlage für die Spezifikation der Daten, die der Generator verarbeiten soll.

Die §21-KHEntgG-Daten bilden im Kontext des virtuellen Gesundheitssystems die Behandlungsfälle nach und werden durch Dateien im CSV-Format (Comma Separated Value) repräsentiert. Es beschreibt den Aufbau einer Textdatei zur Speicherung oder zum Austausch einfach strukturierter Daten. Im Dokument *Merkblatt zum Verfahren der Datenlieferung nach §21-KHEntgG*, Kapitel 4 werden alle CSV-Dateien aufgeführt, die von Krankenhäusern, auf §21 KHEntgG zutreffend, an *3M Medica* übermittelt werden. Desweiteren werden Dateiformat, Struktur des Datensatzes, Inhalte der Datenfelder und jährliche Änderungen der Datenformate für Datenjahre vereinbart. Im Dokument *Vereinbarung nach §21 KHEntgG - Anlage* werden der Aufbau und die Bedeutung einzelner Datenfelder beschrieben. Das Dokument Fehlerverfahren für die Datenerhebung behandelt Datenprüfungen, Stornierung der Datenlieferung und Maßnahmen im Fehlerfall. Aus diesen Dokumenten können für den Generator folgende relevante Informationen festgehalten werden.

1. Die Daten liegen nur im CSV-Format vor.
2. Die Datensätze werden durch das Trennzeichen Semikolon getrennt. Das letzte Feld des Datensatzes schließt ohne Semikolon ab.
3. Spaltenanzahl eines Datensatzes = Semikolonanzahl + 1.
4. Die Bezeichnung der Datenfelder kann sich von Jahr zu Jahr ändern.
5. Jedes Datenfeld hat eine Felddefinition. Die relevanten Definitionen werden im Folgenden gelistet. Weitere Details können aus der Prüftabelle im Dokument *Fehlerverfahren* entnommen werden.
    a. Muss/Kann: Muss-Felder dürfen nicht leer sein.
    b. Typ: Datentyp (numerisch oder alphanumerisch).
    c. Wertebereich (Werteliste, Minimum und Maximum, Anzahl Nachkommastellen).
6. Die erste Zeile der Datei enthält die Feldbezeichner.

Die in *Merkblatt zum Verfahren der Datenlieferung nach §21-KHEntgG* aufgeführten CSV-Dateien werden in Datengruppen Krankenhauskopf, Falldaten, Ausbildung, Kostenmodul, Kosten und Abrechnung zusammengefasst. Die Gruppen werden in der Tabelle im Dokument Fehlerverfahren für die Datenerhebung samt CSV-Dateien dargestellt. Im Rahmen dieser Arbeit sind jedoch nicht alle CSV-Dateien von Bedeutung. Die relevanten CSV-Dateien sind INFO, KRANKENHAUS, FALL, ICD, OPS FAB und ENTGELTE. Die Dateien INFO und KRANKENHAUS gehören der Datengruppe Krankenhauskopf an. Diese Datengruppe gibt Informationen über die Versorgungseinrichtung. In der Literatur wird dafür auch der Begriff Strukturdaten

verwendet. Die übrigen Dateien bilden die Datengruppe Falldaten, die medizinische Daten eines Behandlungsfalls repräsentiert. Dieses Datenumfeld liefert die Grundlage für ein Datenmodell, das zur Realisation des Generators verwendet wird.

2.4. Generatorumfeld

Zunächst muss verdeutlicht werden um welches System es sich handelt. Aus der Zielsetzung in Kapitel 1.1 ist ersichtlich, dass die Funktionalität als Webservice zur Verfügung stehen soll und dass die Konfiguration über ein Benutzerinterface zu erfolgen hat. Literatur- und Onlinerecherchen haben eine Vielzahl an Technologien ergeben, die im Zusammenspiel mit einer Entwicklungsumgebung der Konzeption und der Entwicklung des Generators verwendet werden können. Im Kontext der Vielzahl an Technologien wird ein Sonderkapitel 3 Technologien eingeführt, das in Frage kommende Technologien näher beschreibt.

Des Weiteren können aus der Zielsetzung noch folgende Fragen abgeleitet werden, die als Grundlage für einen Ansatz zur Realisierung dienen können.

1. Welche Art von Daten wird als Input und Output für den Generator definiert?
2. Ist eine Vorverarbeitung der Daten notwendig?
3. Müssen Daten zwecks Wiederverwendung persistent gehalten werden?
4. Was ist unter Simulation von Daten zu verstehen und welche Voraussetzungen müssen die Daten erfüllen?

Diese Fragen werden in den nächsten Abschnitten angeschnitten und in Kapitel 5 Anforderungsanalyse näher spezifiziert. Sie sollen den Entwickler dabei unterstützen richtige Entscheidungen im Bezug auf die Architektur der Webanwendung treffen zu können. Vorab wird in der Abbildung 3 eine Übersicht über den Generator gegeben, die bereits das Input und Output des Generators spezifiziert.

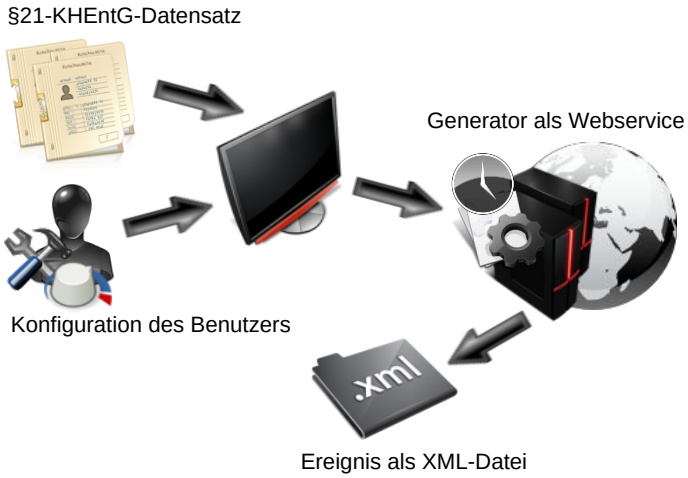


Abbildung 3 Komponenten des Generators

Das Input des Generators besteht aus zwei Komponenten. Die Komponente Datenimport wird durch den §21-KHEntgG-Datensatz spezifiziert. Die zweite Komponente spezifiziert die Konfiguration des Benutzers. Der Benutzer operiert mit seinen Eingaben auf den Daten, die durch den §21-KHEntgG-Datensatz repräsentiert werden. Genauer gesagt geht es um Ereignistypen, die der Benutzer anlegt und die mit Behandlungsfalldaten verknüpft werden. Durch das Anlegen einer Stichprobe werden die Behandlungsfallereignisse bzw. Fallereignisse weiter spezifiziert. Mit einer Zeitkomponente werden die Ereignisse intern im Generator nacheinander abgearbeitet. Als Ergebnis wird eine XML-Datei geschrieben. Die Zeitkomponente und das Schreiben der XML-Dateien werden als Simulationskomponente, die letztendlich die Kernfunktionalität darstellt, definiert.

Die Frage nach der Vorverarbeitung der Daten hängt davon ab, wie gut die Kommunikationspartner zusammenarbeiten. Das Ziel eines virtuellen Gesundheitssystems ist unter Anderem auch die Interoperabilität zu veranschaulichen. Die Interoperabilität impliziert mehrere Integrationsformen. Das vereinbarte CSV-Datenformat zwischen Krankenhaus und Datenstelle *3M Medica* veranschaulicht beispielsweise die Datenintegration. Die Daten werden im Krankenhaus aufgenommen und können von der Datenstelle 3M Medica ausgewertet werden, weil dort das CSV-Datenformat ebenfalls bekannt ist. Neben der Datenintegration gibt es weitere Integrationsformen, die in (Lehmann 2004 S.584) behandelt werden. Im Hinblick auf das Output des Generators, welches in Kapitel 1.1 das XML-Format spezifiziert, muss der Generator eine Konvertierung der Daten vornehmen.

In der Regel hat ein Patient mehrere Ereignisse, die durch Ereignistypen repräsentiert werden. Ein einfaches Beispiel für Ereignistypen sind Aufnahme oder Entlassung. Aus §21 -Daten und Ereignistypen werden Fallereignisse konstruiert. Durch die Definition einer Stichprobe werden die Fallereignisse zu ereignisbasierten Nachrichtenprofilen spezifiziert. Die ereignisbasierten Nachrichtenprofile werden zeitlich simuliert. Die Möglichkeit, mehrere Stichproben aus dem Datenbestand zu kreieren, erfordert eine dauerhafte Speicherung der Behandlungsereignisse.

Unabhängig vom Ereignistyp muss jedes Fallereignis einen Zeitstempel aufweisen, um zeitlich in einen Zusammenhang gebracht werden zu können. Des Weiteren spielen noch Parameter eine Rolle, durch die ein Fallereignis je nach Ereignistyp weiter spezifiziert werden kann. Die Datenfelder Aufnahmedatum, OPS-Datum, OPS-Code und Entlassungsdatum sind potentielle Datenfelder zur Spezifizierung von Ereignistypen. Das Datenfeld Aufnahmedatum spezifiziert den Ereignistyp Aufnahme. Die Datenfelder OPS-Datum und OPS-Code spezifizieren den Ereignistyp OP, denn das Datenfeld OPS-Code beschreibt eine medizinische Tätigkeit und alle OPS-Codes im fünften Kapitel des OPS-Katalogs spezifizieren jeweils eine Operation. Das Datenfeld Entlassungsdatum spezifiziert die Entlassung des Patienten aus dem Krankenhaus. Zur Simulation werden die Behandlungsereignisse zeitlich in einen Zusammenhang gebracht. Die Abbildung 4 beschreibt eine mögliche Ereigniskette, die in einem Behandlungsfall auftreten kann.

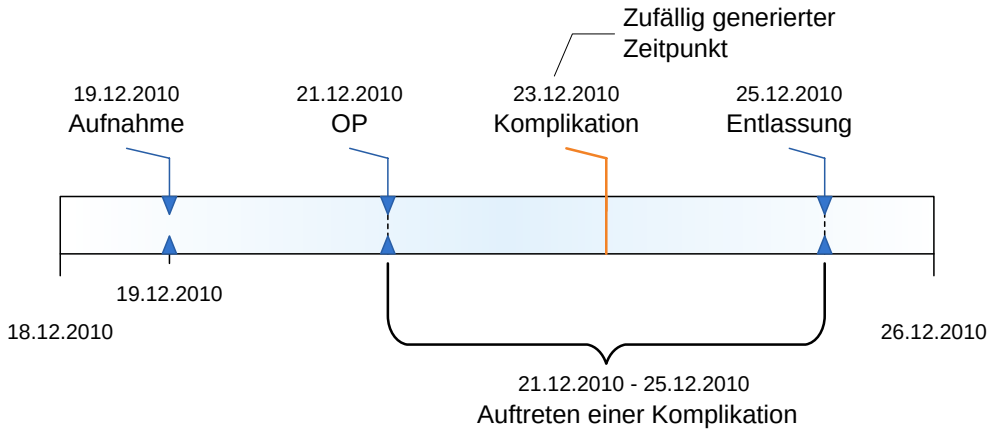


Abbildung 4 Zeitpunkt für ein wahrscheinlichkeitsbasiertes Ereignis

Neben den zeitlich gesteuerten Ereignissen können Ereignisse auch unerwartet auftreten. Solche Ereignisse werden als wahrscheinlichkeitsbasierte Ereignisse bezeichnet. Ein Beispiel für ein wahrscheinlichkeitsbasiertes Ereignis ist das

Auftreten einer Komplikation während des Krankenhausaufenthaltes. Potenzielle Datenfelder für Komplikationen sind Diagnoseart und Sekundärcode. Diagnoseart sagt aus, ob eine Hauptdiagnose oder Nebendiagnose vorliegt. Das Datenfeld Sekundärcode wird als Nebendiagnose behandelt und beschreibt eine Komplikation. Eine Komplikation kann theoretisch nur innerhalb einer Krankenhausbehandlung auftreten – nach der Aufnahme und vor der Entlassung. Es ist jedoch problematisch wahrscheinlichkeitsbasierte Ereignisse auf einem Zeitstrahl abzubilden, weil für sie kein Datum existiert. Ein möglicher Ansatz wahrscheinlichkeitsbasierte Ereignisse in den Griff zu bekommen, ist das Betrachten der Zeitspanne um das wahrscheinlichkeitsbasierte Ereignis herum. Die Zeitspanne wird in der Abbildung 4 durch den Klammerausdruck dargestellt. Grundsätzlich ist die Wahrscheinlichkeit für eine Komplikation nach einer OP gegeben. In dem Fall sind zwei Zeitpunkte von Bedeutung, zum einen die letzte OP und zum anderen die Entlassung. Der OP-Zeitpunkt kann durch das Datenfeld OPS-Datum ermittelt werden. Der OPS-Code gibt Auskunft über das operierte Organ. Das Datenfeld Sekundärcode liefert Informationen über das von Komplikationen betroffene Organ. Der für den Zeitstrahl benötigte Zeitpunkt kann mit Hilfe eines Zufallsgenerators aus Zeiten der letzten OP und der Entlassung gewonnen werden, siehe Abbildung 4. Wahrscheinlichkeitsbasierte Ereignisse sind also verschiedenen Bedingungen unterworfen und somit schwerer zu handhaben.

Neben den zeitgesteuerten und wahrscheinlichkeitsbasierten Ereignissen ist die Geschwindigkeit einer Simulation von Bedeutung. Je nach zur Verfügung stehender Computerhardware läuft eine Simulation bei gleicher Datenmenge unterschiedlich schnell ab. Das System sollte vor der Simulation analysiert werden. Besonders die Schreibgeschwindigkeit der Festplatte sollte vor der Simulation ermittelt werden. Des Weiteren ist die Gesamtanzahl der Ereignisse von Bedeutung, denn sie definieren jeweils eine XML-Datei. Diese Parameter können durch einen Simulationsfaktor beschrieben werden, mit dem die Zeitraffung für eine Simulation eingestellt werden kann.

Aus den Überlegungen in diesem Kapitel können zunächst folgende Informationen für den Generator gewonnen werden.

1. Der Input des Generators besteht aus zwei Komponenten, zum einen dem Datenimport in CSV-Dateien und zum anderen Benutzereingaben über eine Benutzeroberfläche.

2. Der Generator wird als Webservice konzipiert mit einem Benutzerinterface für Benutzerinteraktionen.
3. Die Ereignisse werden durch Benutzer im Benutzerinterface definiert und mit den §21-Daten verknüpft.
4. Es gibt zwei Ereignisarten, die eine ist zeitgesteuert und die andere wahrscheinlichkeitsbasiert.
5. Das CSV-Format ist bei der Verarbeitung nach XML-Format zu konvertieren.
6. Die Simulation ist mit technischen Parametern zu belegen, die vor der Simulation ermittelt werden müssen.
7. Für mehrere Simulationen müssen die §21-Daten dauerhaft gespeichert werden.
8. Der Simulationsvorgang muss protokolliert werden.

In diesem Kapitel aufgestellten Überlegungen werden in Kapitel 5 fortgeführt. Im nächsten Kapitel werden zum Einen das Umfeld von Internet und Web erläutert und zum Anderen die Arbeitsmittel vorgestellt, die zur Realisierung des Generators als eine Webanwendung in Betracht kommen.

3. Technologien

Dieses Kapitel befasst sich mit technischen Aspekten, die bei der Realisierung des Generators relevant sind. Dabei wird der Schwerpunkt auf Entwicklung von Webanwendungen in Java gelegt. Die Entscheidung, das Web im virtuellen Gesundheitssystem einzusetzen, stützt sich zunächst auf folgende Aspekte. Zunehmende Verwendung der Web-Browser als Frontend fördert die Skalierbarkeit einer Webanwendung. Das bedeutet, dass die Anwendung gleichzeitig mehreren Benutzern zur Verfügung steht. Im Vergleich zu einer konventionellen Desktop-Anwendung reduziert die Installation einer Webanwendung auf einem Server den Pflegeaufwand, weil sie zentral gewartet wird.

Im Zusammenhang mit einem Webservice können Funktionen oder Dienste einer Anwendung zur entfernten Verarbeitung von Daten zur Verfügung gestellt werden, wie es beispielsweise im Telemedizinbereich zur Anwendung kommt. Telematische Anwendungen unterstützen das Gesundheitswesen im Fall der entfernten Verarbeitung, indem Daten wie Biosignale, Bilder oder Versichertendaten zur entfernten Verarbeitung und zur Rückgabe der Ergebnisse übermittelt werden (Haas 2006 S.10). Ein weiteres Beispiel für verteilte Anwendungen aus dem Gesundheitswesen liefert der *DIMDI*-Thesaurus[1]. Er stellt einen Webservice zur Verfügung, mit dem ICD- oder OPS-Codes zur Kodierung von Diagnosen oder Prozeduren nachgeschlagen werden können.

Zunehmende Emanzipation der Patienten führt dazu, dass das Web vor einem Arzttermin zur Informationsbeschaffung beansprucht wird. Aus einem Artikel der Ärzte Zeitung geht hervor, dass über 69% der Deutschen mittlerweile online sind, so dass Arztpraxen das Internet zunehmend als Kommunikationsmedium ansehen (Höhl 2010). Die angeführten Beispiele veranschaulichen, dass das Internet und Web von allen Akteuren im Gesundheitswesen aktiv genutzt werden.

Bevor die technischen Mittel zur Realisierung des Generators als Webanwendung im Einzelnen vorgestellt werden, werden grundlegende Begriffe wie Internet, Web und Webservice näher erläutert. Anschließend werden Webtechnologien inklusive relevanter Frameworks vorgestellt, die zur Realisierung des Generators beitragen können.

[1] DIMDI-Thesaurus: Metathesaurus und semantisches Netzwerk medizinischer Begriffssysteme werden zur Katalogisierung und Indexierung von Medienbeständen eingesetzt (*DIMDI* - Thesauri).

3.1. Internet und Web

Das Internet definiert ein Maschennetz aus Netzwerken, die über geeignete Verbindungseinrichtungen miteinander kommunizieren. Zur Abstraktion wird das klassische Client-Server-Architekturmodell herangezogen. Die Akteure des Internets sind Client und Server, ihre Interaktion beruht auf einem einfachen Schema, siehe Abbildung 5.

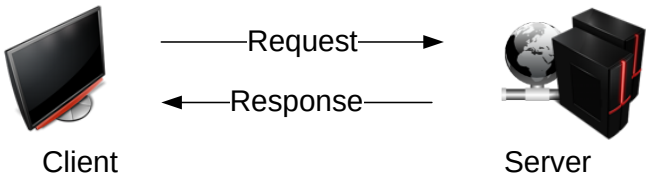


Abbildung 5 Client/Server-Modell

Der Client schickt eine Anfrage an den Server und der Server schickt nach der Bearbeitung der Anfrage eine Antwort. Die Kommunikation zwischen Client und Server wird in einem OSI-Referenzmodell (Open-Systems-Interconnect-Reference-Model) durch Protokolle vereinbart. Insgesamt sind sieben Schichten definiert, die für den Transport von Daten zuständig sind. Jede Schicht hat eine Schnittstelle und eine entsprechende Aufgabe. Die unterste Schicht ist die Physikalische Schicht und wird durch eine physikalische Leitung definiert. Das Signal wird bitweise übertragen. Die Sicherungsschicht sorgt für das Framen der Basisdaten sowie Fehlerbehandlung aus der Bitübertragungsschicht. Die Vermittlungsschicht sucht die passende Route für die zu transportierenden Daten. Dafür wird das Internet-Protokoll (IP) verwendet. Die Transportschicht rundet über das Transmission-Control-Protokoll (TCP) die Aufgaben der transportorientierten Schichten eins bis vier des OSI-Referenzmodells ab. Darüber liegenden Schichten fünf bis sieben sind anwendungsorientiert. Die oberste Schicht ist die Anwendungsschicht, sie repräsentiert in der Regel die Schnittstelle zum Benutzer. Das HTTP-Protokoll ist hier angesiedelt und dient beispielsweise der Darstellung von HTML-Inhalten (Hypertext-Markup-Language).

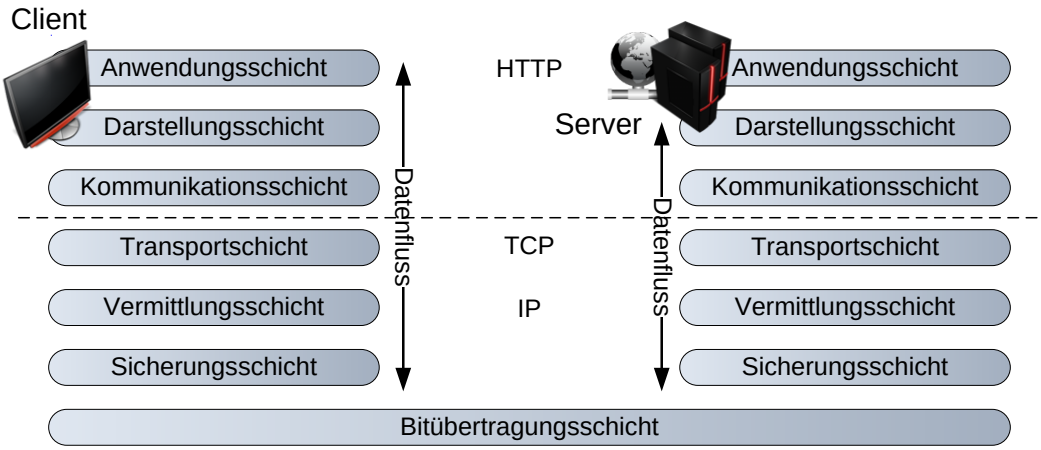


Abbildung 6 OSI-Referenzmodell im Überblick

Die Abbildung 6 stellt die klassische Client-Server-Architektur im Zusammenhang mit dem OSI-Referenzmodell dar. Weitere Informationen über das OSI-Referenzmodell können in (Harnisch 2007 S.13) und (Schreiner 2009 S.4) nachgelesen werden. Kurze Definitionen über das Internet und Web, sowie zeitliche Eckpunkte in der Entwicklung sind in (Bauer 2008 S.1) zu finden.

Das Web repräsentiert eine Ansammlung von Dokumenten und Rechner-Anwendungen im Internet. Mit Auszeichnungssprachen, wie beispielsweise HTML, werden Dokumente im Internet als Webseiten dargestellt. Jede Webseite kann durch Referenzen mit anderen Webseiten verknüpft werden. Für die Darstellung der Webseiten wird ein Webbrowser benötigt, der in der Regel als Client fungiert.

Das aktuelle Web wird durch den Begriff Web 2.0 repräsentiert. Es beschreibt einen Wandel des Internets in der Hinsicht, dass die Entwicklung von Webinhalten personalisiert wird. Das bedeutet, dass der Benutzer beispielsweise eigene Beiträge in Blogs oder auf Seiten von Wikipedia erstellen kann. Bekannte Kennzeichen von Web 2.0 sind unter Anderem Ajax[2] (Asynchronous Java Script and XML) und Webservices. In (Behrendt & Zeppenfeld 2008 S.10) wird Web 2.0 ausführlich in Theorie und Praxis behandelt.

Aus Internet und Web lässt sich der Begriff Webanwendung wie folgt definieren. Eine Webanwendung ist eine Software, die auf Basis von Internet-Technologien und Protokollen arbeitet (Bauer 2008 S.4).

[2] Ajax: Ein Konzept der asynchronen Datenübertragung zwischen einem Browser (Client) und einem Server.

3.2. Webservice und SOA

Die Definition des WWWC (World Wide Web Consortium) beschreibt einen Webservice als eine Softwareanwendung, die durch eine eindeutige URI identifizierbar ist und deren Schnittstelle als XML-Artefakt definiert, beschrieben und gefunden werden kann. Desweiteren unterstützt ein Webservice die direkte Interaktion mit anderen Softwareagenten durch XML-basierte Nachrichten, die über Netzwerkprotokolle ausgetauscht werden (Johner & Haas 2009 S.45). Übertragen auf das Client-Server-Modell ist ein Webservice eine Server-Anwendung, die über XML mit einem Client kommuniziert. Webservices werden häufig in Zusammenhang mit SOA (Service-Orientierte-Architektur) gebracht. SOA verfolgt das Ziel aus der Menge der zur Verfügung stehender Services oder Dienste neue Anwendungen zu kreieren. Die Zusammenstellung der Dienste orientiert sich dabei in der Regel an Geschäftsprozessen, die in einem Unternehmen ablaufen. Die Komponenten des Generators sind in der Hinsicht Dienste, die gemeinsam den Geschäftsprozess – Simulation von nachrichtenbasierten Ereignissen – bilden. Dabei gilt zu unterscheiden, dass diese Dienste nicht mit Webservices gleichzusetzen sind. Ein Dienst, wie beispielsweise das Auslesen von §21-Daten zur Konvertierung nach XML, kann auch innerhalb einer bestimmten Funktionalität oder Methode, in dem Fall Datenimport ausgeführt werden, die nach außen als Webservice zur Verfügung steht. Webservices sind keine Voraussetzung für SOA, sie sind aber ein Sprachmittel um SOA zu definieren. Die Herausforderung dabei ist die Granularität der Dienste. Feine Granularität verursacht einen Overhead hinsichtlich Integration weiterer Softwareagenten und erschwert das Testen. Grobe Granularität verursacht das Problem, dass ein Geschäftsprozess nicht spezifisch genug abgebildet werden kann (Johner & Haas 2009 S.47). Im Zusammenhang mit der Generatorentwicklung wird der Ansatz verfolgt den Geschäftsprozess des Generators als Webservice zu realisieren. Dazu wird in Kapitel 3.4.3 ein passendes Framework oder genauer gesagt eine API vorgestellt.

Neben Webservices beschreiben entfernte Prozeduraufrufe (RPC) oder entfernte Methodenaufrufe (RMI) eine ähnliche Technologie, die in den verteilten Anwendungen zum Einsatz kommt. Der Schwerpunkt wird grundsätzlich auf Webservices gelegt, daher werden hinsichtlich der Übertragung von Daten über HTTP zwei Ansätze betrachtet. Zum Einen geht es um Webservices, die das SOAP-Protokoll einsetzen und zum Anderen gibt es REST-basierte Webservices.

3.2.1.SOAP und WSDL

Das Simple Object Access Protocol (SOAP) ist ein Nachrichtenprotokoll für den Austausch von Daten in XML. Zur Beschreibung der Datenformate, die aufgrund der Heterogenität entstehen, wird XML-Schema herangezogen. Das bedeutet, wenn eine Methode in einem anderen Programm aufgerufen wird, werden Parameter und Rückgabewerte immer als XML-Fragmente verschickt (Scholz & Niedermeier 2009 S.499). Zur Übertragung von XML-Nachrichten wird in der Regel HTTP aus dem OSI-Referenzmodell eingesetzt, siehe Abbildung 6. Die Nutzung von XML und XML-Schema lehnt sich an den Begriff Binding an, der in Kapitel 3.4.1 näher erläutert wird. Der Aufbau einer SOAP-Nachricht wird durch die Teile Envelope, Header und Body definiert. Die Wurzel des XML-Dokuments wird durch das Envelope-Element beschrieben. Das Kind des Envelope-Elements muss ein Body-Element sein, das die eigentlichen Nutzdaten trägt. Optional kann zuvor ein Header-Element stehen, in dem Metainformationen zum Routing stehen können (Heuser & Holubek 2009 S.98).

Die Definition der Schnittstelle erfolgt mit Web Service Description Language (WSDL). Ein WSDL-Dokument beschreibt den Aufbau der Webservices mit allen Funktionen und Übergabe-Parametern. Außerdem gehört die Unterstützung der Protokolle sowie Erreichbarkeit des Webservices ebenfalls zu den Informationen des WSDL-Dokuments (Scholz & Niedermeier 2009 S.500).

3.2.2.REST und WADL

Der Begriff Representational State Transfer (REST) ist eine Alternative zu SOAP und wird eher als ein Architekturstil definiert (Heuser & Holubek 2009 S.367). Dieser Architekturstil orientiert sich an Ressourcen, die durch Daten oder Funktionen definiert werden. Prinzipiell wird REST durch folgende Kernelemente definiert. Eine Ressource ist über URI (Uniform Resource Identifier) identifizierbar. In der Regel handelt es sich dabei um eine Webseite mit einem Link. Der Austausch von Ressourcen erfolgt durch das zustandslose HTTP-Protokoll. Es stellt Standardmethoden (GET, POST, PUT, DELETE) zur Verfügung, mit denen die Ressource manipuliert werden kann. Je nach Anforderungen an Ressourcen werden sie durch unterschiedliche Formate wie HTML oder XML repräsentiert.

Weiterführende Informationen zu REST können in Literatur (STEFAN 2009) und (Richardson & Ruby 2007) nachgelesen werden.

Für die Erstellung von Webservices in REST existiert ebenfalls eine Programmierschnittstelle, Java API for RESTful Web Services. Analog zu WSDL werden in REST Dateien in Web Application Description Language (WADL) definiert. WADL beschreibt die Ressourcen und Formate in XML. Im Gegensatz zu WSDL ist WADL jedoch lediglich auf das HTTP-Protokoll zugeschnitten (STEFAN 2009 S.145).

3.3. Webtechnologien

Die Entwicklung von Softwareanwendungen beruht auf folgende Prinzipien. Durch den Einsatz von Architekturmitteln kann eine Softwareanwendung optimal strukturiert werden. Der Einsatz von Entwurfsmustern und Frameworks verhilft dem Entwickler wiederverwendbare Strukturen effizient und fehlerminimierend zu entwickeln. Als Referenz dafür wird für die Entwicklung einer Webanwendung die im Allgemeinen bekannte Java-EE-Plattform näher betrachtet. Die wichtigsten Konzepte der Java-EE-Plattform werden durch Komponenten und Container repräsentiert. Servlets und Java Server Pages (JSP) sind Beispiele für Java-EE-Komponenten und werden durch Webcontainer[3] ausgeführt (Turau u. a. 2004 S.1). Im Rahmen der Diplomarbeit werden lediglich die Komponenten vorgestellt, die bei der Realisierung des Generators relevant sind.

3.3.1. JSP und Servlet

Java Server Pages und Servlets definieren die wichtigsten Java-EE-Komponenten. Darauf baut das Java-EE-Modell auf, das durch eine drei-Schichten-Architektur definiert wird. Die mittlere Schicht beschreibt dabei die Anwendungslogik und entkoppelt so den Client vom Server. JSPs sind ausschließlich zur Präsentation von

[3] Webcontainer: In Literatur wird ein Webcontainer häufig auch als Servletcontainer bezeichnet. Apache Tomcat ist ein Beispiel für einen Servletcontainer.

Anfrageergebnissen zuständig. Sie implementieren weder die Vermittlungsschicht noch die Anwendungslogik und erlauben damit eine bessere Trennung zwischen Präsentation und der Anwendungslogik. Im Allgemeinen definiert JSP eine Web-Programmiersprache zur einfachen dynamischen Erzeugung von HTML- und XML-Ausgaben eines Webservers (Turau u. a. 2004 S.3).

Servlets nehmen Anfragen aus dem Webbrowser entgegen und delegieren sie nach der Analyse der Anfragedaten an eine entsprechende Komponente. Der Zusammenhang zwischen JSP und Servlet wird in Abbildung 7 verdeutlicht.

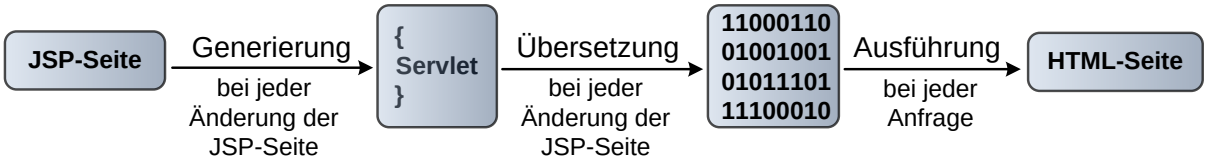


Abbildung 7 JSP-Technologie (Turau u. a. 2004 S.26)

JSPs werden vor der ersten Anfrage oder bei jeder Änderung der JSP-Seite in Servlets übersetzt. Bei weiteren Anfragen der JSP-Seite führt der Web-Container automatisch das generierte und übersetzte Servlet aus. Das Ergebnis wird in HTML im Webbrowser dargestellt (Turau u. a. 2004 S.26). In der Regel enthält eine JSP-Seite außer HTML-Code weitere Codefragmente wie Scriplets, Deklarationen oder Ausdrücke. Das macht die JSP-Seite unübersichtlich und kompliziert. Durch die Auslagerung der Codefragmente in Tags wird JSP zu einer Template-Sprache, die lediglich zur Generierung visueller Darstellung dient (Saake & Sattler 2003 S.311). Die Tags werden durch die JSP-Standard-Library (JSTL) spezifiziert. Neben der JSTL gehört die Expression Language (EL) ebenfalls zum Kern der JSP-Technologie. Die EL ermöglicht Scripting-Elemente zu programmieren und zusammen mit JSTL soll die Entwicklung der JSP-Seiten erleichtert werden.

3.3.2.HTML

Die Hyper Text Markup Language ist bereits als Auszeichnungssprache erwähnt worden. Sie dient der Strukturierung von Webinhalten und wird bei der Entwicklung sowie auch Darstellung verwendet. Neben HTML gibt es Weiterentwicklungen wie CSS (Cascading Style Sheet), XML oder Ajax. Sie erlauben eine Trennung unterschiedlicher Aspekte von Webinhalten: Während HTML die Inhalte an sich verkörpert, beschreibt CSS, wie diese Inhalte dargestellt werden sollen (Münz 2006 S.53). CSS lässt sich in HTML integrieren, es empfiehlt sich jedoch zwecks der Übersichtlichkeit im Code das Layout in eine gesonderte CSS-Datei auszulagern. Der Begriff Ajax wird hinsichtlich der Erweiterungen des Generators relevant. Grundsätzlich beschreibt das Konzept Ajax eine asynchrone Kommunikation zwischen Client und Server und sorgt dafür, dass lediglich der aktualisierte Seiteninhalt neu aufgebaut wird.

3.3.3.XML

Extensible Markup Language ist ebenfalls eine Auszeichnungssprache und dient zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien. Zur Strukturierung werden wie bei HTML Tags und Attribute verwendet, jedoch im Gegensatz zu HTML ist die Bedeutung einzelner Tags nicht festgelegt, sondern individuelle Tags und Attribute können festgelegt werden. Mit XML-Schema ist es möglich zum einen XML-Daten inhaltlich sowie strukturell einzuschränken und zum anderen XML-Daten zu validieren. Ein XML-Parser parst XML-Daten und kann ein XML-Schema zur Validierung nutzen. Der Begriff Parsen bezeichnet das Einlesen eines XML-Dokuments gemäß der XML-Syntax in den Speicher einer Anwendung. Das Gegenteil zum Parsen ist der Serialisierungsprozess, der die XML-Daten aus einer Anwendung beispielsweise in Form einer Datei ausgibt. Zur Verarbeitung von XML-Daten gibt es zwei Verarbeitungsweisen – streambasiert und modellbasiert. Sie werden in der Tabelle 1 dargestellt.

Streambasiert	Modellbasiert
•SAX: Simple API for XML. SAX durchläuft XML-Dokumente sequentiell und interpretiert dabei die XML-Syntax. Für jeden gefunden XML-Baustein wird ein Ereignis - SAX-Event genannt - erzeugt. SAX nutzt das Prinzip des Push-Parsing. •StAX: Streaming API for XML. StAX arbeitet wie SAX jedoch basiert auf dem Prinzip des Pull-Parsing. Neben Parsen der XML-Dokumente ist mit StAX Serialisieren möglich.	•DOM: Document Object Model. DOM ist ein generisches Objektmodell zur Darstellung aller möglichen XML-Dokumente. •Binding: Ein Konzept, das sich von der dokumentnahen XML-Verarbeitung verabschiedet. Das Modell des Binding-Konzeptes wird an eine spezifische Grammatik angepasst. Die Klassen werden anhand der Grammatik auf ein XML-Dokument abgebildet.

Tabelle 1 Verarbeitungsarten von XML-Daten (Scholz & Niedermeier 2009 S.80)

In der streambasierten Verarbeitung wird zwischen Push- und Pull-Parsing unterschieden. Das Push-Parsing wird durch den SAX-Parser vertreten. Hierbei wird der Programmfluss durch den Parser über das Auslösen von Ereignissen gesteuert. Der Entwickler stellt eine Handler- oder Callback-Klasse zur Verfügung und der Parser liefert schrittweise die Inhalte. Jeder Aufruf der Callback-Klasse liefert nur die aktuellen Knoteninformationen, nicht jedoch den Baumkontext (Scholz & Niedermeier 2009 S.78). Die Alternative bietet der StAX-Parser, er parst XML-Daten nach dem Pull-Prinzip. Die Steuerung des Parsens wird hier durch die Anwendung übernommen. Mit StAX werden XML-Dokumente ähnlich geparst wie bei SAX, allerdings nicht char- oder byteweise, sondern in typischen XML-Bausteinen, wie Elementen, Attributen und Text (Scholz & Niedermeier 2009 S.83). Die Unterschiede zwischen SAX und StAX werden in Abbildung 8 veranschaulicht.

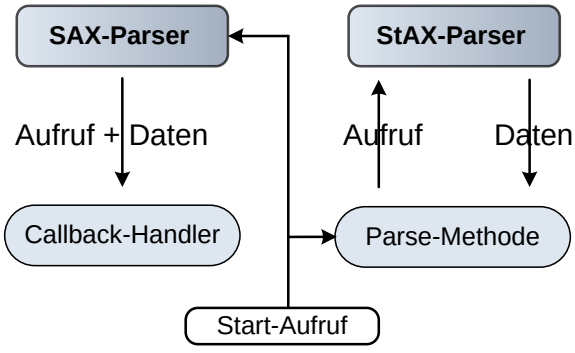


Abbildung 8 Unterschiede SAX und StAX (Scholz & Niedermeier 2009 S.82)

Bei der Verwendung von StAX stehen zwei prinzipiell unterschiedliche Programmierschnittstellen zur Verfügung: die Cursor-API und die Event-Iterator-API. Mit Hilfe der Cursor-API wird das XML-Dokument in Blöcke aufgeteilt und ein Cursor von einem Block zum nächsten bewegt. Die Variante mit der Event-Iterator-API funktioniert identisch mit dem Unterschied, dass die Informationen über den aktuellen Knoten in ein passendes XML-Event-Objekt verpackt werden.

In der modellbasierten XML-Verarbeitung werden mit DOM alle Inhalte eines XML-Dokumentes auf Java-Objekte abgebildet. Hierfür wird eine Schnittstelle bereitgestellt, die den Zugriff auf die Bausteintypen eines XML-Dokuments regelt. Zur Veranschaulichung wird als Beispiel die nach XML konvertierte FALL-Datei als DOM-Modell dargestellt.

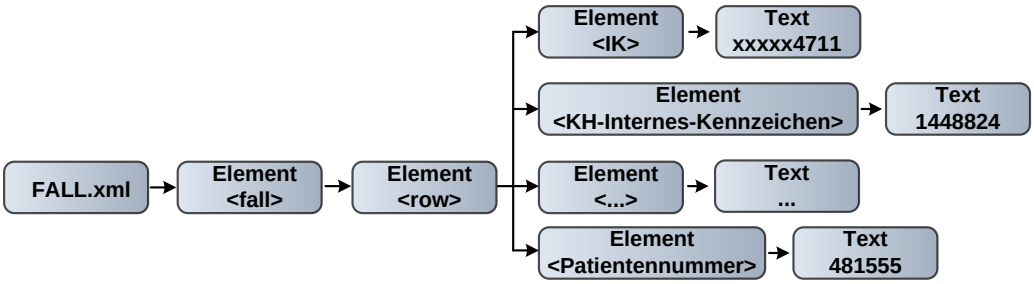


Abbildung 9 FALL.xml als Objektmodell

Die Objekte aus der Abbildung 9 verweisen über Referenzen aufeinander und bilden somit die logische XML-Struktur eins zu eins ab.

Alle drei vorgestellten Technologien zur XML-Verarbeitung arbeiten auf der untersten Ebene direkt mit den XML-Konstrukten wie Elementen, Attributen oder Text. Sie bieten zwar die meiste Flexibilität, allerdings wird der Verarbeitungscode dafür recht umfangreich. Als Alternative ist es mit Binding möglich sich von der dokumentnahen Verarbeitung zu lösen und somit ein spezifisches Objektmodell aufzubauen. Das Binding beschreibt die Umwandlung der XML-Dokumente in Java-Instanzen und umgekehrt. Eine mögliche Umsetzung dazu bietet JAXB, das in Kapitel 3.4.1 näher vorgestellt wird.

Im Rahmen dieser Arbeit wird die XML-Verarbeitung eine bedeutende Rolle einnehmen, weil es in mehreren Bereichen eingesetzt wird. Die vorgestellten Parser und weitere Details zur XML-Verarbeitung können in der Literatur (Scholz & Niedermeier 2009) nachgelesen werden. Im Bezug auf den Generator wird der

Schwerpunkt auf Binding gelegt, weil das Zielformat der Fallereignisse in XML definiert wird.

3.4. Frameworks und APIs

Der Begriff Framework wird im Zusammenhang mit Strukturierung und Wiederverwendung von Software verwendet. Ein Framework definiert eine Grundstruktur zur Entwicklung von Komponenten. Es gibt keine Vorgaben hinsichtlich der technologischen Umsetzung. Aufgrund der großen Anzahl an Technologien – dasselbe gilt für Frameworks – werden in diesem Unterkapitel relevante Frameworks, hinsichtlich der Entwicklung, zur Betrachtung ausgewählt. In Kapitel 4.1.1 werden diesbezüglich die Alternativen betrachtet.

3.4.1.JAXB

Java Architecture for XML Binding (JAXB) ist eine Programmierschnittstelle in Java. Der Begriff Binding definiert Verarbeitung der XML-Daten in Java, jedoch hebt es sich von der dokumentnahen XML-Verarbeitung, wie mit SAX oder DOM ab. Die Relevanz dieser Technologie spiegelt sich in folgenden Aspekten wider.

1. Bidirektionale Bindung
2. Unterstützung von Webservices
3. Validierung

Die bidirektionale Bindung definiert sowohl das Generieren der Java-Klassen aus einem XML-Schema als auch den umgekehrten Weg. Der Weg vom XML-Schema nach Java-Klassen wird zur Generierung des Datenmodells für §21-Daten angewendet. Näheres dazu siehe Kapitel 7.1.

Im Fall, dass eine Anwendung als Webservice veröffentlicht wird, muss das existierende Datenmodell der Anwendung durch ein WSDL-Dokument beschrieben werden. JAXB automatisiert dieses Vorgehen, indem aus dem Datenmodell ein XML-

Schema generiert wird, das bei der Kommunikation über XML und SOAP verwendet wird. Es wird in der Jax-WS-Technologie als Standard eingesetzt (Michaelis & Schmiesing 2006 S.3).

Die nach XML konvertierte §21-Daten können mit JAXB gegen das XML-Schema validiert werden. Sowohl beim Parsen der XML-Daten in eine Objektstruktur (Unmarshalling), als auch beim Serialisieren der Java-Objekte in eine XML-Struktur (Marshalling) ist es möglich anhand des XML-Schemas zu validieren (Michaelis & Schmiesing 2006 S.3).

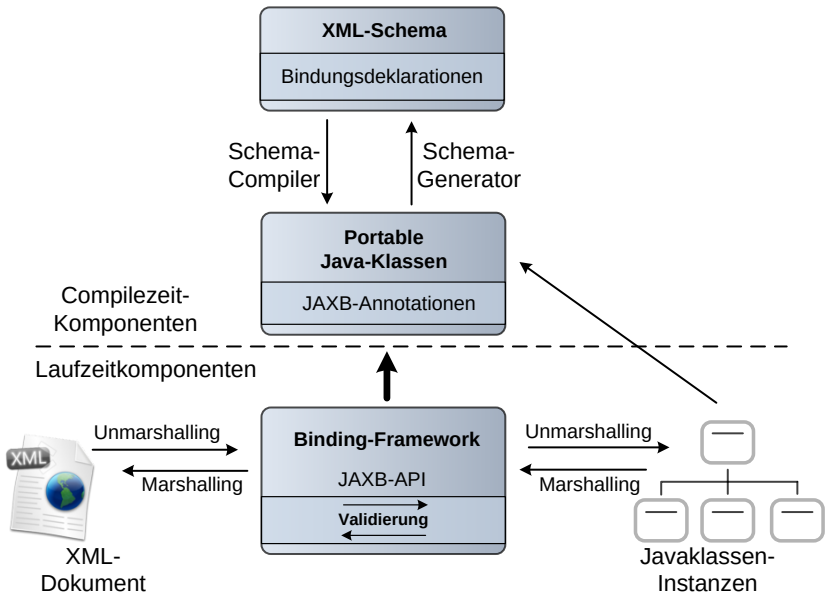


Abbildung 10 JAXB-Architektur (Michaelis & Schmiesing 2006 S.6)

Die Abbildung 10 beschreibt zwei Bereiche der JAXB-Architektur. Das Datenmodell für die §21-Daten wird zur Entwicklungszeit mit Hilfe von XML-Schema und Schema-Compiler generiert. Das Parsen der §21-Daten und somit der Transfer der XML-Daten in eine Objektstruktur wird zur Laufzeit durch das Binding-Framework realisiert.

Als alternative Binding-Technologie bietet sich das Framework Castor an, welches wie JAXB in Spring ebenfalls unterstützt. Für beide Technologien existiert im Springframework die Bibliothek OXM (Object-XML Mapping), die die Marshalling- und Unmarshalling-Prozesse Framework-spezifisch abstrahiert.

3.4.2.JDO

Java Data Objects (JDO) ist eine Spezifikation mit dem Ziel die Persistenz-API zu standardisieren und somit ein Verfahren zur transparenten Speicherung von Java-Objekten bereitzustellen. Dieses Ziel wird unter Anderem durch zwei Aspekte verfolgt. Zum Einen durch die Anforderungen, dass jede Java-Klasse persistenzfähig sein soll und zum Anderen durch die Objektidentität. Dabei wird in JDO die Objektidentität auf drei Arten definiert. Ausführliche Informationen darüber können in der Literatur (Turau u. a. 2004 S.283) nachgelesen werden. Im Gegensatz zu Object/Relational-Mappern (O/R-Mapper) ist es mit JDO möglich neben relationalen Datenbanken auch objektorientierte Datenbanken, LDAP-Server oder XML-basierte Lösungen zu verwenden (Wolff 2009 S.195). Im Bereich der Persistenz-Technologien ist JDO somit eine Alternative zur nativen JDBC-Technologie (Java Database Connectivity) sowie zu den Mapping-Frameworks wie zum Beispiel JPA (Java Persistence API) oder Hibernate. Im Zusammenhang mit dem Generator sollten mehrere Simulationen möglich sein. Die Behandlungsfälle, so wie sie in 2.3 Datenumfeld aus den §21-Daten definiert werden, werden für die Verknüpfung an Ereignisse verwendet. Aus diesem Grund wird JDO im Zusammenhang mit objektorientierten Speicherung der Behandlungsfälle in Betracht gezogen.

Im Mittelpunkt von JDO stehen Klassen, welche die Objekte des Anwendungsbereichs modellieren. Diese sind stark vernetzt und bilden ein Objektgeflecht. In der Entwicklung einer JDO-Anwendung oder einer JDO-Persistenzschicht werden in der Regel folgende Schritte aus (Turau u. a. 2004 S.288) notwendig.

1. Für Datenobjekte werden Java-Klassen erstellt und compiliert. Dabei steht die Modellierung der Semantik im Vordergrund. Trotzdem sollte berücksichtigt werden, dass ein zu komplexes Modell die Performanz negativ beeinflusst.
2. Eine weitere Eigenschaft im JDO-Konzept ist der Enhancement-Prozess. Dafür und für den Abbildungsprozess der Klassen im Datenspeicher werden Meta-Informationen benötigt. Zu den Meta-Informationen gehören unter Anderem folgende Überlegungen. Es ist festzustellen welche Klassen und Instanzvariablen persistent sein müssen, welche Instanzvariable den Primärschlüssel bildet und welche Daten beim Start geladen werden. Die Meta-Informationen werden in der Regel in einer XML-Datei abgelegt und tragen die Dateiendung *jdo*. Alternativ zur XML-Datei ist es laut des JDO-

Tutorials (JDO-Tutorial) und der JDO-3-Spezifikation (Java Data Objects 3 - Maintenance Release 3) möglich Meta-Informationen direkt im Code per Annotationen unterzubringen.

3. Der Enhancement-Prozess legt die Meta-Informationen zugrunde und macht die compilierten Java-Klassen persistenzfähig, indem der Java-Bytecode modifiziert wird.
4. In diesem Schritt wird der Datenspeicher vorbereitet. Dazu werden im Fall eines relationalen Datenspeichers ein Datenbankschema generiert und entsprechende Tabellen angelegt.

JDO folgt ähnlichen Prinzipien wie JDBC oder sonstige O/R-Mapper. Für den Zugriff auf Funktionalitäten von JDO wird, wie bei anderen Persistenz-Technologien, eine Referenz auf einen Persistenz-Manager benötigt, der bei einer Persistenz-Manager-Factory angefordert werden kann. Außerdem ist es mit JDO-Query-Language möglich gezielt Objekte aus dem Datenspeicher zu selektieren. Eine Anfrage mit JDOQL benötigt grundsätzlich drei Angaben – Menge der JDO-Objekte aus denen selektiert wird, Klasse der Objekte aus denen selektiert wird und eine Filterbedingung. Das Laden der selektierten Objekte erfolgt erst beim Zugriff innerhalb einer Iteration über die Antwortmenge.

JDO wird genauso wie andere Persistenz-Technologien von Spring unterstützt. Dafür werden in Spring entsprechende Interfaces zum Handling von Persistenz- und Transaktionen angeboten, die das Arbeiten mit JDO etwas erleichtern.

3.4.3.Jax-WS

Java API for XML Web Services (Jax-WS) ist eine Programmierschnittstelle für Web Services. Es handelt sich dabei um Standard-Java-Technologie zur Erstellung verteilter interoperabler Software-Anwendungen (Scholz & Niedermeier 2009 S.495). Die Akteure der Servicewelt sind in der Regel Vertragspartner. Ein Service-Provider bietet Dienste an und ein Service-Consumer nimmt die Dienste in Anspruch. Für einen problemlos aufrufbaren Webservice müssen Vereinbarungen getroffen werden. Die Erstellung von Webservices mit Jax-WS beruht dabei auf zwei Ansätzen. Der Contract-First-Ansatz definiert die Vereinbarung der Schnittstelle mit WSDL. Das WSDL-Dokument repräsentiert die technischen Aspekte eines Webservices.

Anschließend wird mit Jax-WS ein Codeskelett generiert, in dem der Entwickler seine Geschäftslogik implementieren kann. Dieser Ansatz setzt umfangreiches Wissen und Erfahrung voraus um möglichst fehlerfrei eine Schnittstelle mit WSDL zu beschreiben. Aus diesem Grund wird dieser Ansatz nur am Rande erwähnt. Ausführliche Informationen zum Contract-First-Ansatz können in der Literatur (Scholz & Niedermeier 2009) nachgelesen werden. Die Alternative oder der etwas weniger komplexe Weg wird durch den Contract-Last-Ansatz beschrieben, den Jax-WS ebenfalls unterstützt. In dem Fall fängt der Entwickler wie gewohnt mit Implementierung der Java-Klassen an. In der Regel sind dafür aus (Scholz & Niedermeier 2009 S.514) folgende Schritte zu tun.

1. Serviceklasse schreiben als Webservice. Jax-WS stellt dafür entsprechende Annotationen zur Verfügung. Die Eine mit der Annotation *@Webservice* definierte Java-Klasse oder genauer gesagt ein Interface wird als Webservice veröffentlicht.
2. Für die Bereitstellung der Serviceklassen als Webservice benötigt Jax-WS Hilfsklassen. Genauer gesagt geht es dabei um JAXB-Klassen, die Java-Objekte nach XML-Nachrichten überführen, um von aufrufenden bzw. an aufrufende Methoden oder Operationen gesendet werden zu können. Mit diesem Schritt wird aus Model- und Serviceklassen inklusive Annotationen das WSDL-Dokument generiert.
3. Jax-WS liefert einen eigenen http-Server mit, mit dem in diesem Schritt der bereitgestellte Webservice getestet werden kann.
4. Die Bereitstellung mehrerer Webservices erfolgt durch das Deployment auf dem Anwendungsserver.

Das genaue Vorgehen zum Contract-Last-Ansatz wird in der Literatur (Scholz & Niedermeier 2009 S.514) ebenfalls beschrieben. Im Hinblick auf die Entwicklung des Generators bietet Jax-WS mit beiden Ansätzen eine Hybrid-Lösung an, mit der die Anwendungslogik des Generators als Webservice abgebildet werden kann. Im Zusammenhang mit dem Framework Spring wird Jax-WS durch die Schnittstelle *SimpleJaxWsServiceExporter* integriert.

3.4.4.Spring

Das Framework Spring verfolgt den Ansatz, die Komplexität der bis dato zur Verfügung stehenden APIs zu reduzieren. In (Liebhart 2007 S.74) wird Spring als Meta-Framework definiert, das eine einheitliche Zugriffsschicht auf technische Details, realisierende Frameworks und Libraries bereitstellt. Demnach wird es nicht nur zur Entwicklung von Webanwendungen eingesetzt. Zur Vereinheitlichung der APIs bietet Spring entsprechende Module an. Hinsichtlich der Relevanz wird die Auswahl in der Abbildung 11 vorgestellt.

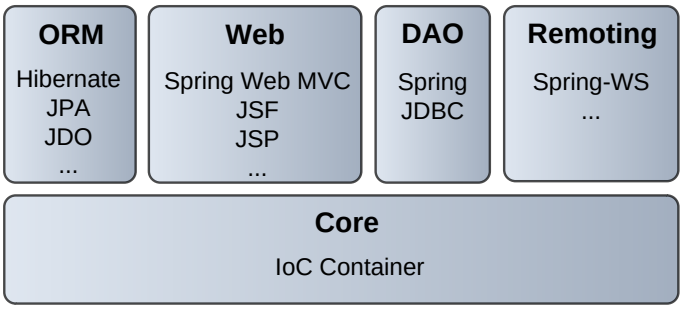


Abbildung 11 Auswahl der Spring-Module

Das Core-Modul aus Abbildung 11 stellt grundsätzliche Funktionalität des Spring-Frameworks bereit. Dazu gehört Erstellung, Konfiguration und Verwaltung von Beans[4] (Walls & Breidenbach 2007 S.10). Mit Hilfe von *Dependency Injection* werden mit diesem Modul Beans verschaltet, die letztendlich eine Anwendung bilden. Die Erstellung von Beans erfolgt entweder über Bean-Factory oder Anwendungskontext. Letzteres sorgt für mehr Leistungsfähigkeit bei der Nutzung von Spring. Zur Veröffentlichung muss die erstellte Bean in einer Konfigurationsdatei deklariert werden. In der Regel werden Beans in XML-Dokumenten definiert. In einem Podcast von Jax-TV (8 Möglichkeiten, Projekte mit Spring aufzubauen) gibt der Autor des Literaturwerkes Spring 3 (Wolff 2009) eine umfangreiche Übersicht, wie ein Projekt mit Spring aufgebaut werden kann. Die Verschaltung oder auch als Injektion bekannt, kann entweder durch Konstruktoren oder durch Getter- und Setter-Methoden geschehen. Für weitere Informationen zum Bean-Management wird auf (Walls & Breidenbach 2007 S.37) und (Wolff 2009 S.21) verwiesen.

Die Auswahl der Spring-Module außer dem Core-Modul spiegelt sich in der Java-EE-Plattform wider. Hinsichtlich der Datenschicht der Java-EE-Plattform – in Spring durch

[4] Beans: Im Zusammenhang mit Spring handelt es sich hier um POJOs (Plain Old Java Objects).

Persistenz-Module definiert – stellt Spring die Module ORM (Object Relational Mapper) und DAO (Data Access Object) zur Verfügung. Das ORM-Modul abstrahiert sämtliche Objekt-relationale-Mapper, indem es dafür Schnittstellen anbietet. In dem Fall werden für JDO – siehe Kapitel 3.4.2 – die Schnittstellen *JdoDaoSupport* zur Ressourcenverwaltung und *JdoTransactionManager* zum Transaktionsmanagement zur Verfügung gestellt. Das DAO-Modul abstrahiert den nativen Ressourcenzugriff per JDBC, indem es Codevereinfachungen durchführt. Dazu werden ebenfalls Schnittstellen wie *JdbcTemplate* und *JdbcDaoSupport* zur Verfügung gestellt. In beiden Modulen wird das DAO-Entwurfsmuster eingesetzt. Es regelt den Zugriff auf Datenobjekte, die von der Serviceschicht an die DAO-Schicht delegiert werden. Weitere Informationen zu den beiden Modulen werden ausführlich in (Oates & Bachlmayer 2008 S.119; Walls & Breidenbach 2007 S.150; Wolff 2009 S.164) beschrieben.

Die Serviceschicht liegt hinsichtlich der Java-EE-Plattform über der Datenschicht. Genauer gesagt ist es die Anwendungslogik. Die Anwendungslogik einer Webanwendung verteilt zur Verfügung zu stellen wird durch das Remoting-Modul unterstützt. Im Remoting-Modul vereint Spring neben Webservices auch konventionelle Remote-Procedure-Call-Modelle. Sie werden in der Literatur (Walls & Breidenbach 2007 S.278) samt Technologie ausführlich behandelt. Im Bezug auf Webservices stellt Spring die Komponente Spring-WS zur Verfügung. Es ist das Pendant zur in 3.4.3 Jax-WS vorgestellten Programmierschnittstelle mit dem Unterschied, dass es den Contract-First-Ansatz bei Erstellung von Webservices bevorzugt. Es vereinfacht jedoch die Schnittstellendefinition, indem die Schnittstelle in XML-Schema definiert wird und das WSDL-Dokument durch Spring-WS automatisch generiert wird.

Die Präsentationsschicht schließt die Drei-Schichten-Architektur nach dem Java-EE-Modell ab. Im Hinblick auf die Umsetzung des Generator-Projektes wird lediglich die Komponente Spring-MVC herangezogen. Spring-MVC verwendet das MVC-Entwurfsmuster, das aus drei Komponenten – Model, View und Controller (MVC) – besteht. Das Model hält die Daten, der Controller implementiert die Logik zur Verarbeitung der Anfragen aus dem Browser und die View zeigt das Ergebnis an. Die Abbildung 12 aus (Wolff 2009 S.233) veranschaulicht die MVC-Beziehungen im Web-Container.

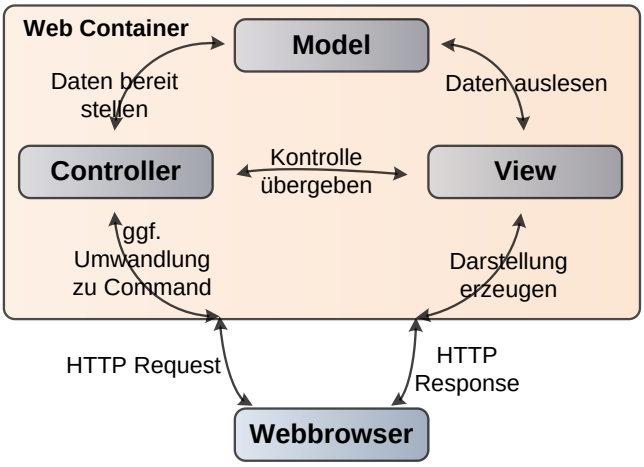


Abbildung 12 MVC in Spring

Neben den MVC-Komponenten gibt es weitere Akteure oder Spring-Beans, die an der Verarbeitung eines Requests beteiligt sind. Das *DispatcherServlet* ist die Anlaufstelle eines Requests. Für die Suche nach einem passenden Controller wird das Handler-Mapping bemüht. Nach der Verarbeitung durch den Controller werden die Daten im Model bereitgestellt. Das Model wird in der ModelAndView-Datenstruktur um die View ergänzt und als Ergebnis geliefert. Die View wird durch den *ViewResolver* bereitgestellt. Der etwas komplexere Vorgang bei der Verarbeitung eines Requests, durch Einsatz von Validatoren, sowie des Command-Entwurfsmusters als Datenbehälter, wird in (Wolff 2009 S.274) ausführlich beschrieben. Spring-MVC wird in der Generatorentwicklung als Webinterface eingesetzt.

3.5. Auswahl der Technologien

Abschließend zum Kapitel 3 wird in diesem Unterkapitel aus theoretischer Sicht die Auswahl der Technologien begründet. In Kapitel 3.4 wurden hinsichtlich der Drei-Schichten-Architektur nach dem Java-EE-Modell für jede Schicht entsprechende Frameworks vorgestellt. Zum Teil wurde ansatzweise die Vorauswahl für die Entwicklung Generators angedeutet. Auf den Punkt gebracht sind folgende Technologien für die Entwicklung des Generators relevant. Die Technologien werden dabei jeweils einer Schicht zugeordnet. Neben Spring existieren weitere Frameworks, mit denen Webanwendungen erstellt werden können. Abschließend zu diesem

Kapitel wird das Spring-Framework dem GWT- (Google Web Toolkit) und JSF-Framework (Java Server Faces) gegenübergestellt.

3.5.1.Datenschicht

Die Datenschicht wird durch Domainobjekte und Datenzugriffsobjekte repräsentiert. Im Allgemeinen beschreiben die Domain-Objekte eine Abbildung zwischen den Daten im Backend und denen zur Laufzeit. Im Kapitel 6.1 werden Domainobjekte des Generators in Form von Klassen näher vorgestellt. Die Datenzugriffsobjekte beschreiben den Zugriff auf Daten und werden in der Regel durch das DAO-Entwurfsmuster definiert. Die Funktionalitäten von JAXB ermöglichen die Generierung der §21-Domainklassen, die für den Zugriff auf XML-Dateien eingesetzt werden. Des Weiteren kann JAXB zur Validierung eingesetzt werden. Nähere Details zur Generierung von Domainklassen sowie das Parsen von XML-Daten werden im Implementierungsansatz aufgeführt. Im Hinblick auf Webservices definiert JAXB den Kern von Jax-WS um Java-Objekte nach XML und wieder zurück zu transformieren.

JDO wird hinsichtlich Erweiterbarkeit und Abstraktion des Datenzugriffs interessant. Es bietet breite Unterstützung für Datenbanken hinsichtlich der Art der Speicherung sowie hinsichtlich der Abbildung der Daten auf das Backend. Zur Laufzeit erstellte Behandlungsfallereignisse stehen lediglich während der Konfiguration zur Verfügung. Zur späteren Änderung müssen sie persistent abgelegt werden. Die Entscheidung eine objektorientierte Datenbank als Backend einzusetzen basiert auf die Tatsache, dass ausschließlich mit XML und Java-Objekten gearbeitet wird und eine relationale Datenbank zusätzlich Aufwand generiert. Als Objektdatenbank wird db4o in Betracht gezogen. Sie steht als Open-Source-Lizenz zur Verfügung und lässt das Persistieren der Objekte über JDO zu. Im Hinblick auf Erweiterungen ist es mit JDO möglich auch eine relationale Datenbank einzuführen.

3.5.2.Serviceschicht

Losgelöst von verteilten Anwendungen wird in der Serviceschicht die verarbeitende Logik des Generators untergebracht. Dafür müssen lediglich Schnittstellen definiert werden. Im Fall, dass der Generator als Webservice zur Verfügung gestellt wird, müssen diese Schnittstellen nach dem Contract-Last-Ansatz mit Annotationen aus Jax-WS versehen werden. Jax-WS bietet zwei Ansätze zur Erstellung von Webservices, damit bietet es gute Voraussetzungen hinsichtlich einer möglichen späteren Änderung von Contract-Last nach Contract-First, profiliert. Der Contract-Last-Ansatz ist in der Regel einfacher, weil der Entwickler sich nicht mit der Komplexität von WSDL beschäftigen muss, sondern wie gewohnt mit der Implementierung anfangen kann. Für die Abbildung der Funktionalitäten des Generators als Webservice wird das Framework Jax-WS ausgewählt.

3.5.3.Präsentationsschicht

Für das Webinterface wird die Spring-MVC-Komponente (Model View Controller) ausgewählt. Der Einsatz des MVC-Entwurfsmusters begünstigt die Modularisierung, Flexibilität und Austauschbarkeit der Anwendung. Die View-Komponente unterstützt zahlreiche Technologien, die in (Walls & Breidenbach 2007 S.479) vorgestellt werden. Neben Java Server Pages werden JSFs unterstützt. Letzteres wird in Kapitel 3.5.5 der Spring-MVC-Komponente gegenübergestellt. In der View werden JSPs im Zusammenhang mit JSTL und EL, siehe Kapitel 3.3.1, eingesetzt.

3.5.4.Abgrenzung zu GWT

Das Google Web Toolkit ist ein Framework zur Entwicklung von Ajax-basierten Webanwendungen. Der Begriff Ajax wurde bereits im Zusammenhang mit Web 2.0 in Kapitel 3.1 erwähnt. Im Grunde vereint Ajax die schon bekannten Technologien HTML, XML, CSS und Java Script und definiert somit die Dynamik, die sich durch das asynchrone Verhalten kennzeichnet. Den Kern von Ajax bildet das

Kommunikationsobjekt *XMLHttpRequest,* auf das bei jeder asynchronen Anforderung der Daten mit Java Script zugegriffen wird (Steyer 2007 S.15). Im Zusammenhang mit GWT wird das Erstellen Ajax-basierter Webanwendungen hinsichtlich der Komplexität reduziert (Steyer 2007 S.20). In Literatur (Steyer 2007) und (Hanson & Tacy 2007) wird das Vorgehen zur Erstellung einer Webanwendung anhand einiger Beispiele beschrieben.

An dieser Stelle werden Gemeinsamkeiten und Unterschiede von GWT mit Spring aufgeführt. Im Gegensatz zu Spring konzentriert sich GWT lediglich auf die Entwicklung von Webanwendungen. Spring unterstützt zahlreiche Konfigurationsmöglichkeiten und ist dadurch nicht nur für Webanwendungen geeignet. Desweiteren unterstützt es zahlreiche Entwurfsmuster. GWT-Webanwendungen zeigen durch den Einsatz von Ajax eine gute Performanz im Browser. Diesbezüglich ist Spring-MVC im Nachteil, weil es Ajax direkt nicht unterstützt. Über Direct Web Remoting (DWR) ist es jedoch möglich Spring-MVC Ajax-fähig zu gestalten. Beide Frameworks – GWT und Spring – lassen sich vereinen. Ein Beispiel wird in Literatur (Dwyer 2008 S.63) demonstriert, in dem Fall eignet sich jedoch der Einsatz von GWT zur Einbindung bestimmter Widgets[5]. Im Zusammenhang mit der Java-EE-Plattform lässt sich die Drei-Schichten-Architektur mit beiden Frameworks umsetzen. In GWT wird jedoch die Struktur strikt nach Client und Server getrennt, somit wird die Anwendungslogik in Schnittstellen und die dazugehörigen Implementierungen aufgeteilt. Dieses Konzept wird durch das RPC-Modell definiert, indem die Service-Schnittstellen Client-seitig und die Implementierung Server-seitig definiert werden. Aus dieser Tatsache folgt der nächste Aspekt hinsichtlich Remoting. Während Spring mehrere Ansätze zum Remoting zur Verfügung stellt, ist das Grundkonzept von GWT auf das RPC-Modell beschränkt. Das RPC-Modell – so wie in der Literatur (McLaughlin u. a. 2006) definiert – weist aufgrund der Heterogenität der Daten Probleme mit der Repräsentation auf (McLaughlin u. a. 2006 S.306). Dieses Interoperabilitätsproblem wird in GWT durch Einsatz von Java Script Object Notation[6] (JSON) beseitigt. JSON ist jedoch nicht XML und kann somit nicht zur Entwicklung von SOAP-Webservices herangezogen werden.

[5] Widget:Dynamische, wiederverwendbare Komponenten zur Gestaltung von Oberflächen (Steyer 2007 S.25).

[6] JSON: Ein Nachrichtenformat, welches zur Übermittlung strukturierter Daten eingesetzt wird und Bestandteil von GWT ist.

Grundsätzlich stellen beide Frameworks das Potential zur Realisierung des Generators als Webanwendung zur Verfügung. Im Hinblick auf Interoperabilität und Sprachenunabhängigkeit sind Webservices per SOAP essentiell und müssen bei der Auswahl der Technologien berücksichtigt werden.

3.5.5. Abgrenzung zu JSF

Java Server Faces ist ein Framework zur Entwicklung von Benutzerschnittstellen für eine Java-EE-Anwendung bzw. als Teil davon (B. Müller 2006 S.10). Im Zusammenhang mit Spring ist es unter Anderem die Alternative zu Spring-MVC. JSF orientiert sich an der traditionellen GUI-Programmierung, indem es auf der Oberfläche Komponenten platziert und sie im Hintergrund mit Java-Beans verschaltet (Wolff 2009 S.284). Die Verbindungen zwischen Komponenten der Oberfläche und den Java-Beans werden durch die EL hergestellt, die konventionell bei JSPs eingesetzt wird. Im Gegensatz zu Spring-MVC stellt JSF eine Event-Verarbeitung mit vier Eventarten zu Verfügung (B. Müller 2006 S.91). Die Event-Verarbeitung oder Verarbeitung der Eingaben des Benutzers werden in Spring-MVC in den Controller ausgelagert. Die View in Spring-MVC basiert auf reinem HTML-Code, der über JSTL und EL mit dem Controller verknüpft wird. Zusätzliche Validatoren können die Eingaben des Benutzers überprüfen. In JSF wird die Validierung bereits in den GUI-Elementen deklariert und würde in diesem Fall zu Coderedundanz führen, weil beispielsweise im Controller oder in den im Hintergrund verknüpften Java-Beans von JSF ebenfalls validiert wird. Dieser Aspekt ist bei der Pflege und Wartung besonders wichtig. Sowohl Spring-MVC als auch JSF sind für die Entwicklung der Präsentationsschicht interessant – beide unterstützen das MVC-Entwurfsmuster und beide bieten *Dependency Injection* an. Genau wie GWT lässt auch JSF den Einsatz von Ajax grundsätzlich zu, während Spring-MVC erst über DWR Ajax-fähig wird.

Der Einsatz von Spring-MVC beruht auf konventionellen Technologien wie Servlet, JSP und HTML. Die Konfiguration der Beans, Unterstützung hinsichtlich Remoting oder Datenzugriff macht das Hinzuziehen von JSF überflüssig.

4. Generator-Projekt

In diesem Kapitel werden projektspezifische Gestaltungsbereiche aufgeführt. Dazu wird der Generator aus der Projekt-Perspektive betrachtet. Das Spiralmodell beschreibt eine iterative und inkrementelle Vorgehensweise in einem Softwareentwicklungsprozess. Es enthält vier Quadranten, die in der Abbildung 13 folgende Aktivitäten definieren.

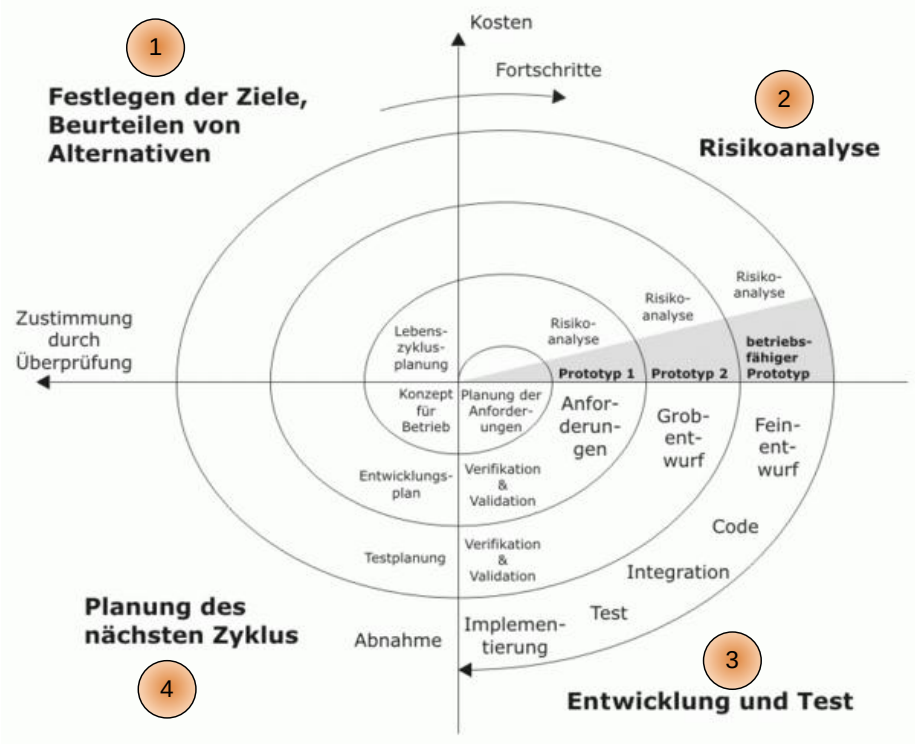


Abbildung 13 Spiralmodell in Anlehnung an (Spiralmodell)

Die Aktivitäten werden wiederholt, bis keine relevanten Risiken das Projektziel gefährden. Weitere Informationen zum Spiralmodell sind in (Pfetzing & Rohde 2009 S.191) zu finden. Hinsichtlich der Konzeption und Entwicklung des Generators werden die üblichen Phasen der Softwareentwicklung zugrunde gelegt. Die Phasen werden durch Anforderungsanalyse, Design und Implementierung und Test definiert und werden in entsprechenden Kapiteln beschrieben. Neben der Iteration der Phasen gehört das Risikomanagement zum Bestandteil des Spiralmodells. Ein Risiko beschreibt die Möglichkeit für das Auftreten eines Schadens im Projekt. In Kapitel 4.2

werden mögliche Risiken und deren Vorbeugung hinsichtlich des Generator-Projektes näher betrachtet.

4.1. Projektziele

Das primäre Ziel des Generators – Simulation ereignisbasierter Nachrichtenprofile – wurde im Kapitel 1.1 definiert. Das virtuelle Gesundheitssystem in Kapitel 2 beschreibt die Beteiligten des Generator-Projektes. Der Gedankenansatz in Kapitel 2.4 beschreibt die grobe Strukturierung des Generators in Komponenten. Für das Projekt notwendige Themengebiete technischer Sicht werden in Kapitel 3 zusammengefasst. Im Hinblick auf das Webservice-Konzept, vorgestellt in Kapitel 3.2, wurden die Funktionalitäten des Generators als ein Geschäftsprozess betrachtet. Die Funktionalitäten oder, aus der SOA-Perspektive gesehen, Dienste, werden zunächst durch Datenimport, Benutzereingaben und Simulation definiert. Die Konfiguration der ereignisbasierten Nachrichtenprofile besteht aus Benutzereingaben und ihrer Verknüpfung mit §21-Daten. Die Simulation der ereignisbasierten Nachrichtenprofile wird durch eine Zeitkomponente und XML-Generierung definiert. Der Geschäftsprozess wird im Einzelnen in Abbildung 14 verdeutlicht.

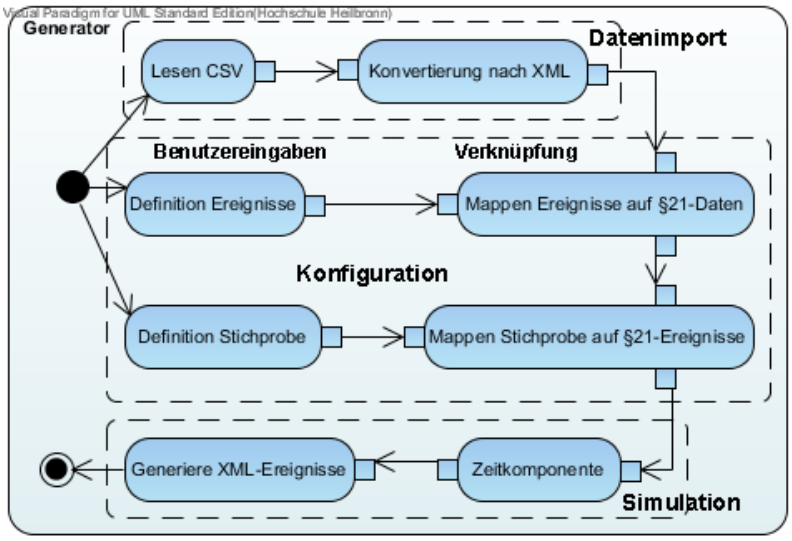


Abbildung 14 Geschäftsprozess des Generators

Das Spiralmodell sieht vor bei der Umsetzung eines Systems mögliche Alternativen heranzuziehen. Dieses Vorgehen schöpft zum einen Ideen hinsichtlich der Realisierung und zum anderen kann es zur Risikominimierung beitragen, wenn Alternativen hinsichtlich des Aufwandes gegenübergestellt werden. Die Komponenten des Generators können aufgrund der Vielzahl von Technologien unterschiedlich realisiert werden. Im Kapitel 4.1.1 werden mögliche Variationen des Generators hinsichtlich der Realisierung untersucht.

Nach der Definition der Ziele werden zunächst Anforderungen an den Generator formuliert und eine grobe Schätzung über die Ressource Zeit gemacht. Die ersten Analysen liefern ein grobes Bild über den Generator und sein Umfeld und werden in Kapitel 5 fortgesetzt. Hinsichtlich der Zeit werden die üblichen Phasen der Softwareentwicklung durch Meilensteine im Zeitplan in der Abbildung 15 wie folgt dargestellt. Die Anforderungsanalyse schließt mit einem Pflichtenheft ab, in dem alle Anforderungen an den Generator aufgeführt werden. Parallel zur Anforderungsanalyse wird der erste Prototyp entwickelt, der zum einen den ersten Eindruck über den Generator vermittelt und zum anderen weitere Ideen oder Änderungspotentiale in die erneute Anforderungsanalyse fließen lässt. Die Prototypenentwicklung ist Bestandteil des Spiralmodells und wird später bei der Risikoanalyse eine besondere Rolle einnehmen.

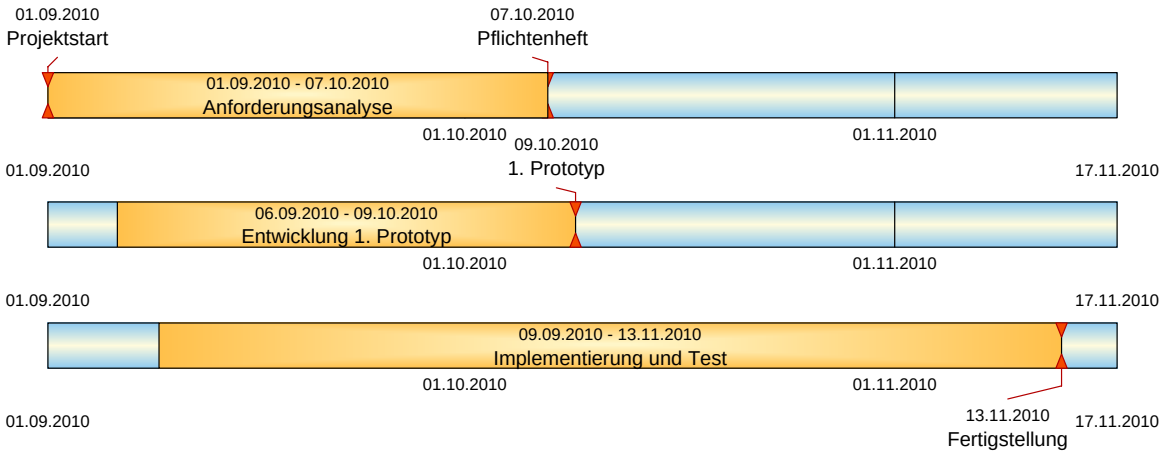


Abbildung 15 Zeitplan zur Konzeption und Realisierung

Die Implementierungs- und Testphase wird durch Ergebnisse aus der Anforderungsanalyse und des ersten Prototypen repräsentiert. Neben

Implementierung und Test wird in dieser Phase das Fachwissen technischer Natur zusammengetragen, um hinsichtlich der Realisierbarkeit Aussagen treffen zu können. Die Erfahrungen mit Technologien fließen wieder in die Anforderungsanalyse und generieren Ideen für mögliche Alternativen.

4.1.1. Alternativen

Zum Spiralmodell gehört unter Anderem das Betrachten der Alternativen. In Kapitel 3 werden die in Frage kommenden Technologien vorgestellt. Als Alternativen werden in diesem Unterkapitel mögliche Ausprägungen des Generators technischer Natur betrachtet, unabhängig davon welche Technologie zum Einsatz kommt. Der Geschäftsprozess aus Kapitel 4.1 definiert Datenimport, Konfiguration und Simulation. Alle drei Komponenten haben die Gemeinsamkeit, dass sie über eine Benutzeroberfläche zugänglich gemacht werden müssen. Die Konfigurationskomponente des Generators definiert Benutzereingaben und deren Verknüpfung auf §21-Daten. Die Simulationskomponente verarbeitet die ereignisbasierten Nachrichtenprofile und schreibt sie als XML-Dateien. Im Zusammenhang mit der Java-EE-Plattform, die eine Drei-Schichten-Architektur definiert, werden Tätigkeiten des Benutzers im Browser, Verarbeitung der Daten sowie dessen Speicherung zunächst durch Daten-, Service- und Präsentationsschicht definiert.

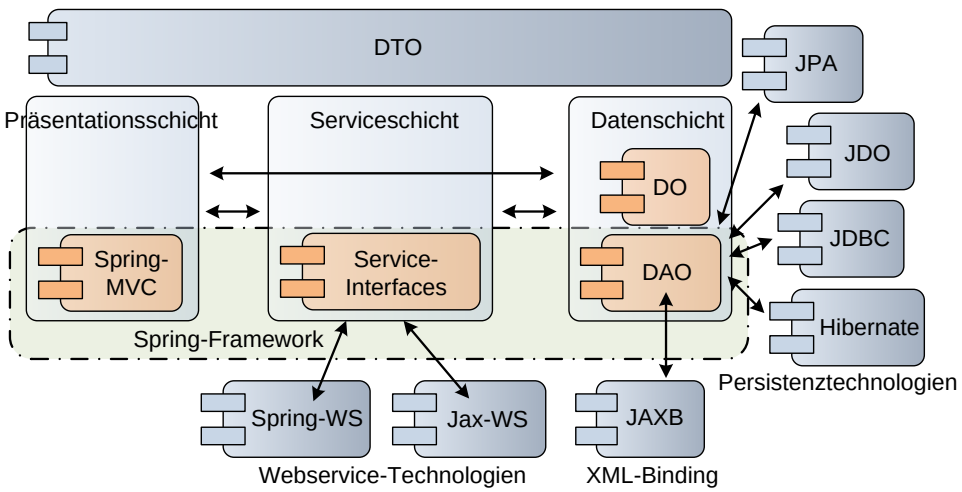


Abbildung 16 Mögliche Architektur des Generators

Die Abbildung 16 beschreibt mehrere Architekturansätze in einem Schaubild. Sie werden im Folgenden im Hinblick auf Technologieeinsatz Schicht für Schicht näher betrachtet.

4.1.1.1. Datenschicht

Die Datenschicht aus Abbildung 16 spezifiziert die DAO-Schicht und die Domain-Schicht. Die DAO-Schicht kümmert sich um den Zugriff auf die Domain-Objekte. Die Domain-Objekte sind in der Regel einfache POJO-Klassen, die Attribute und entsprechende Getter- und Setter-Methoden definieren. Sie werden mit Hilfe einer Persistenz-Technologie auf eine Datenbank abgebildet. An der Stelle ist zu unterscheiden, welche Art von Datenbank verwendet wird. Generell lassen sich vier Arten unterscheiden.

1. Die zu persistierenden Daten werden Stream-basiert als Datei gespeichert.
2. Die Daten werden in einer Objekt-relationalen Datenbank gespeichert.
3. Die Daten werden in einer Objekt-orientierten Datenbank gespeichert.
4. Die Daten werden XML-basiert gespeichert.

Bei der Wahl der Persistenz-Technologie ist zu beachten, in welcher Form die Daten vorliegen. Sind es einzelne Dokumente mit Text oder Daten, die in einer Tabellenstruktur vorliegen? Letzteres entspricht den §21-Daten. Die Benutzereingaben liegen spätestens in der Anfrage als gekapseltes Objekt vor. Hier ist die Entscheidung offen, wie dieses Objekt persistiert wird. Zur Bereitstellung der zu transportierenden Objekte für den Webservice müssen die Objekte von der Objektstruktur in eine XML-Nachrichtenstruktur gebracht werden. Das erfolgt mit der JAXB-Technologie. Insofern liegt der Schwerpunkt bezüglich der Persistenz-Technologien auf den letzen beiden Punkten. Es ist zwar durchaus möglich die §21-Daten zunächst in einer relationale Datenbank abzulegen. Hier ist die Frage nach der Änderbarkeit der Daten entscheidend, denn die §21-Daten liegen aggregiert für ein Jahr vor und werden im Einzelnen nicht geändert, sondern am Stück entweder eingelesen oder entfernt. Ein definiertes Ereignis ist der einzige potenzielle Objekttyp für eine relationale Datenbank, denn eine Stichprobe wird Datei-basiert abgelegt. Aus diesem Grund werden die Persistenz-Technologien JPA, Hibernate und JDBC zunächst zurückgestellt. Im Hinblick auf Erweiterung und Portabilität ist es dennoch

möglich die gewünschte Persistenz-Technologie mit Spring auszutauschen. Durch Einsatz des DAO-Entwurfsmusters ist es möglich ein generisches DAO-Interface zu definieren, das für konkrete DAO-Interfaces erweitert wird. Das Framework Spring bietet für die jeweilige in der Abbildung 16 aufgeführte Persistenz-Technologie ein Template an, das die CRUD-Methoden (Create Read Update Delete) enthält. Daraus folgt, dass die DAO-Schicht je nach Persistenz-Technologie ausgetauscht werden kann. Für das XML-Binding, siehe in Kapitel 3.4.1 wird die JAXB-Technologie verwendet. Sie parst XML-Daten in eine Objektstruktur und serialisiert Objekte nach XML. Dieser Prozess wird bei den SOAP-Webservices oder Webservices verwendet, die intern eine XML-Struktur tragen. In Anbetracht der Tatsache, dass die §21-Daten von CSV nach XML konvertiert werden, wird die JAXB-Technologie verwendet um Behandlungsfälle aus XML-Daten auszulesen und mit Ereignistypen zu verknüpfen.

4.1.1.2. Serviceschicht

Die Serviceschicht kümmert sich um die Verarbeitung der Objekte aus einer Anfrage. Ein Objekt aus Anfrage kann als Beispiel ein Zeichensatz vom Typ String oder ein gekapseltes Objekt sein – bestehend aus Formulardaten, die auf der Oberfläche eingegeben werden. Die Serviceschicht enthält in der Regel Methoden, die etwas berechnen, zuordnen oder zusammenstellen. Die Methoden werden in Interfaces definiert, um so die Implementierung von den Schnittstellen zu entkoppeln. Der Einsatz von Technologien wie Spring-WS oder Jax-WS ermöglicht die Funktionalitäten als Dienste im Web zur Verfügung zu stellen. Die Methoden mit CRUD-Charakter werden üblicherweise in einer DAO-Schicht untergebracht. Es gibt jedoch Ausnahmen – Methoden, die eigentlich der DAO-Schicht angehören, werden als Serviceschicht deklariert. Im Zusammenhang mit dem Generator wird die DAO-Schicht von der Serviceschicht getrennt gehalten.

4.1.1.3. Präsentationsschicht

Die Spring-MVC-Komponente definiert in der Präsentationsschicht weitere Strukturen, die der losen Kopplung dienen. Zur Umsetzung der Präsentationsschicht sind alternativ auch JSF und GWT denkbar. Im Fall GWT muss der Controller von Spring-MVC angepasst werden. Dazu muss der Controller nach dem GWT-RPC-Stil das GWT-spezifische *RemoteServiceServlet* erweitern, um Anfragen entgegen nehmen zu können. JSF unterstützt das MVC-Entwurfsmuster und kann demnach analog zu Spring-MVC eingesetzt werden. Der Controller des Spring-MVC verarbeitet in der Regel Anfragen aus dem Browser. Ein durchdachter Controller leitet die Verarbeitung an eine Service-Schicht weiter. Es gibt jedoch Ausprägungen, bei denen die Controller direkt auf die Datenzugriffsschicht zugreifen.

4.1.1.4. Datentransferobjekt

Das Datentransfer-Objekt (DTO) aus Abbildung 16 ist ein Entwurfsmuster und kann optional beim Design der Serviceschicht eingesetzt werden. An der Stelle ist zu klären, ob die Ergebnisse zum Client als DTOs oder bereits vorhandene Domain-Objekte übertragen werden. In der Regel besitzen Domainobjekte als POJOs lediglich Attribute und Getter- und Setter-Methoden. In einigen Fällen, wie bei der Verwendung von JPA muss in der POJO-Klasse ein leerer Konstruktor definiert sein. Hier besteht die Möglichkeit die Domain-Objekte direkt als Ergebnis für den Client einzusetzen. Oftmals besteht das Ergebnis einer Service-Methode jedoch aus einer Sammlung von Domain-Objekten. In dem Fall eignet sich der Einsatz von DTOs, weil sie weitere Datenstrukturen zur Aggregation der Domain-Objekte enthalten können. Das Ergebnis des Generators ist ein Behandlungsfallereignis, das die Domain-Objekte aus den XML-Daten kapselt. Demnach wäre der Einsatz von Datentransfer-Objekten an der Stelle geeignet. Aufgrund der Datenstruktur der Domain- und Datentransfer-Klassen ist der Einsatz von Datentransfer-Objekten jedoch abzuwägen, weil er hinsichtlich der Pflege und Erweiterung zu Mehraufwand führt (Oates & Bachlmayer 2008 S.155).

4.1.2.Randbedingungen

Zu den Randbedingungen gehört unter Anderem die vollständige Gestaltungsfreiheit im Bezug auf die Technologiewahl. Im Rahmen der vorliegenden Arbeit ist der Zeitrahmen zur Konzeption und Realisierung begrenzt. Als Entwicklungsumgebung wird Spring Source ToolSuite (STS) eingesetzt mit dem Java Development Kit in Version 1.6. Der Einsatz von STS stützt auf die Tatsache, dass sie Spring Source ToolSuite bereits für die Entwicklung mit Spring vorbereitet ist. Alternativ kann die Spring-Umgebung in Eclipse eingebunden werden. Weitere Technologien wurden in Kapitel 3 bereits aufgeführt. Der Einsatz von anderen Hilfsmitteln wird im Pflichtenheft aufgeführt.

4.2. Risikoanalyse

Ein Projekt ist üblicherweise von Risiken umgeben. An mehreren Stellen werden bereits Aspekte aufgeführt, die das Potential für Risiken definieren. Dabei werden zwei Arten von Risiken unterschieden. Zum einen Risiken, die von außen auf ein Projekt einwirken und zum anderen Risiken, die projektintern entstehen können. Die Risiken von außen lassen sich schlecht beherrschen. Ein anschauliches Beispiel für das Generator-Projekt wäre die Abschaffung des §21-KHEntgG oder Änderung des Gesetzes soweit, dass es keinen Dateninput mehr für den Generator ermöglicht. In dem Fall hat die Gesetzgebung allein den Einfluss auf das Projekt. Projektinterne Risiken sind projektbegleitend und werden im Generator-Projekt den im Folgenden erläuterten Risikokategorien zugeordnet.

4.2.1.Risikokategorien

Projektinterne Risiken können in der Regel zu Projektbeginn oder im Laufe des Projektes entstehen. Das Generator-Projekt wird hinsichtlich möglicher Risiken analysiert und dazu werden Risikokategorien aufgestellt. Die drei Kategorien

definieren die Grundlage für mögliche Risiken. Aus ihnen lassen sich weitere Risiken ableiten.

Anforderungen	Technologie	Termin
• Nicht alle Anforderungen erfüllt. • Nachträgliche Änderung in den Anforderungen.	• Freie Technologiewahl. • Falsche Technologiewahl.	• Begrenzter Zeitrahmen.

Tabelle 2 Mögliche Risiken für das Projekt

Die Kategorie Anforderungen beschreibt in Tabelle 2 zwei mögliche Risiken. Die Nichterfüllung von Anforderungen hat in der Regel folgende Ursachen. Die Anforderungen sind komplex oder es stehen nicht genug Ressourcen zur Verfügung um alle möglichen Anforderungen zu erfüllen. Oftmals kommt es zu nachträglichen Änderungen der Anforderungen aufgrund von neuen Ideen oder anderer Betrachtungsweise des Problemgebietes. In beiden Fällen lässt sich das Risiko durch ein Pflichtenheft, und im Zusammenhang mit dem Spiralmodell, durch einen Prototypen minimieren.

Die Kategorie Technologie beschreibt ebenfalls zwei Aspekte. Freie Technologiewahl fördert zwar heterogene Kompositionen eines Systems, es erfordert jedoch Erfahrung auf dem Gebiet um fundierte Entscheidungen über die Technologiewahl treffen zu können. Aufgrund der Entscheidungsfreiheit, der Vielzahl von Technologien und fehlendem Wissen können schnell falsche Entscheidungen hinsichtlich der Technologiewahl getroffen werden.

Die Ressource Zeit bezieht sich auf konkrete Termine – Meilensteine – und steht im Rahmen der Arbeit nur begrenzt zur Verfügung. Diese Kategorie hängt von den anderen beiden ab. Zur Veranschaulichung werden beispielhaft die möglichen Risiken in einer Risikomatrix[7], siehe Abbildung 17, abgebildet.

[7] Risikomatrix dient der Risikosteuerung und wird aus (Ahrendts & Marton 2007 S.26) auf das Generator-Projekt beispielhaft übertragen, um die Beziehung der Risikofaktoren zu veranschaulichen. Die Risikowerte sind Schätzwerte, weil zu der Zeit keine Erfahrungswerte vorliegen.

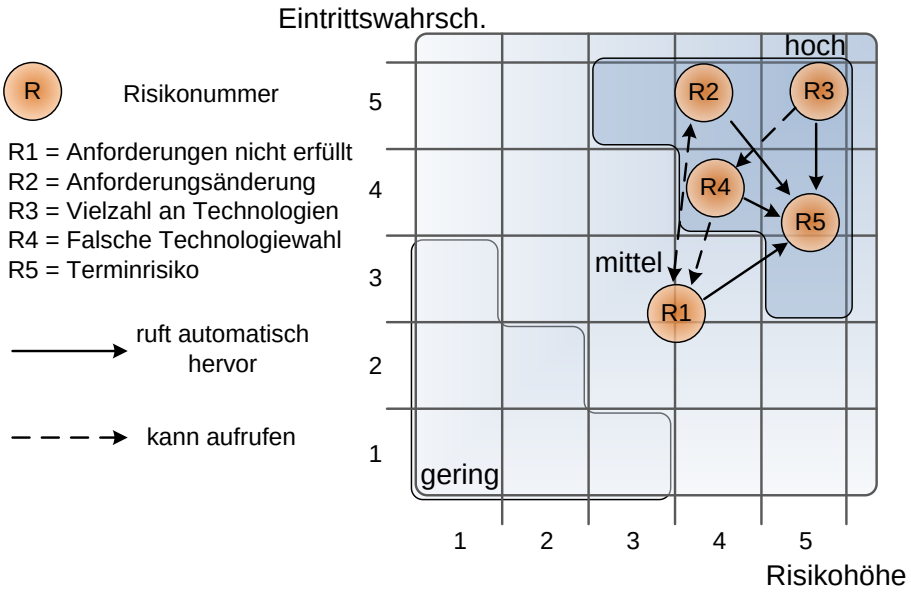


Abbildung 17 Risikomatrix in Anlehnung an (Ahrendts & Marton 2007 S.26)

Die Multiplikation der Risikohöhe mit der Eintrittswahrscheinlichkeit liefert das Schadenausmaß für das Projekt (Ahrendts & Marton 2007 S.25). Im obigen Fall variiert das Maß zwischen 0 und 25. Anhand der ermittelten Größe können die Risiken priorisiert werden. Das höchste Risiko wird durch die Vielzahl an Technologien (R3) und mögliche Anforderungsänderungen (R2) hervorgerufen. Das R3-Risiko gehört zu den Risiken, die bereits vorab geklärt werden müssen. Dessen Nichtbeachtung kann entweder zur falschen Technologieauswahl führen oder viel wahrscheinlicher zur Verzögerung der Termine im Projekt. Das R2-Risiko – Anforderungsänderung – ist neben der Vielzahl an Technologien ein weiteres hohes Risiko und steht mit dem R1-Risiko – Nichterfüllung der Anforderungen – in einer beiderseitigen Beziehung. Nichterfüllung der Anforderungen kann deren Änderung hervorrufen und Änderungen aufgrund neuer Ideen können dazu führen, dass sie aufgrund der Ressource Zeit nicht umgesetzt werden. Alle aufgeführten Risiken führen jedoch dazu, dass Termine nicht eigehalten werden können.

Nach Ermittlung der möglichen Risiken können die vier folgenden Strategien aus (Ahrendts & Marton 2007 S.29) in Betracht gezogen werden.

- Risikoakzeptanz
- Risikoverlagerung
- Risikominderung
- Risikovermeidung

Im Bezug auf das Generator-Projekt sind lediglich die beiden unteren Aspekte relevant, weil die Risikoakzeptanz oder Risikoverlagerung nicht dem Projektziel entsprechen.

4.2.2.Risikostrategien

Risikominderung und Risikovermeidung sind zwei Strategien, die im Generator-Projekt zum Einsatz kommen. Zur Risikominderung gehören Tätigkeiten aus den Kapitel 2 und 3. Sie haben ergeben, dass die Realisierung des Generators mehr Wissen erfordert als zu Anfang des Projektes angenommen wurde. Nähere Details werden in Kapitel 7 aufgeführt. Des Weiteren verzögert sich die Anforderungsanalyse aufgrund weiterer Untersuchungen für die Spezifikation der Ereignistypen sowie Konzeption des Domänenmodells, so dass der Zeitplan und Umfang der Anforderungen angepasst werden müssen. Die Änderungen im Zeitplan werden wie folgt dargestellt.

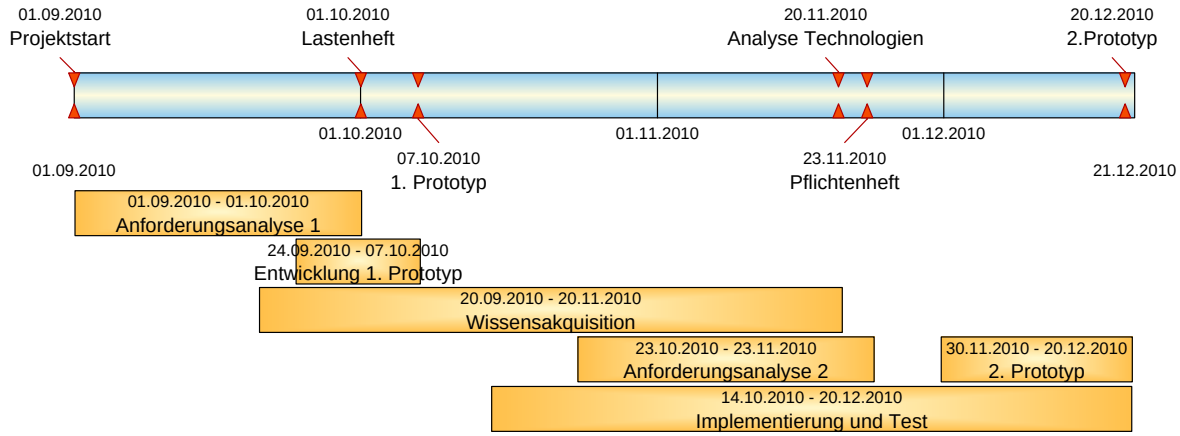


Abbildung 18 Angepasster Zeitplan

Der Zeitrahmen für Konzeption und Entwicklung wird in Abbildung 18 innerhalb der Diplomarbeitsphase verlängert. Die Auseinandersetzung mit Technologien wird der Phase Wissensakquisition zugeordnet. Neue Erkenntnisse in der Anforderungsanalyse führen dazu, dass das Pflichtenheft angepasst werden muss.

Das Anpassen des Zeitplans, der Anforderungen und des Technologieeinsatzes werden aus den aufgestellten Risikokategorien in folgende Beziehung gesetzt, siehe in Abbildung 19.

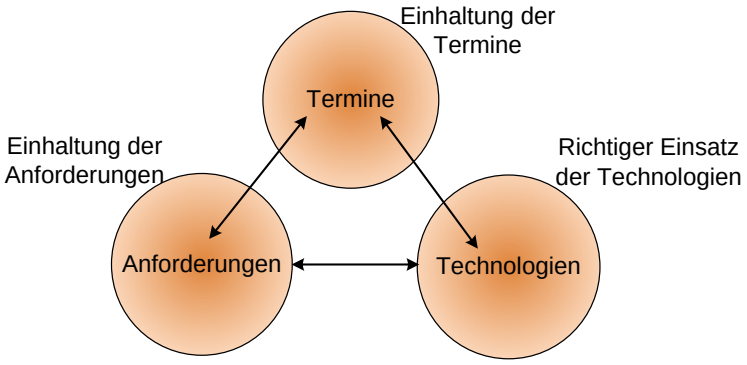


Abbildung 19 Beziehung der Risikokategorien

Die Eckpunkte der Risikokategorien können hinsichtlich der Risikominderung beeinflusst werden. Im Generator-Projekt wird die Diplomarbeitsphase fest vorgegeben. Es steht dem Entwickler und Projektmanager jedoch frei innerhalb dieses Zeitrahmens zu agieren – siehe in Abbildung 18 Angepasster Zeitrahmen. Die Eckpunkte Anforderungen und Technologien können durch Anpassen der Anforderungen und Reduzieren des Technologieeinsatzes ebenfalls variiert werden. In Kapitel 5 werden Anforderungen an den Generator ermittelt. Des Weiteren werden Anforderungen im Zusammenhang mit dem Prototyp priorisiert, um unter Anderem zur Risikovermeidung beizutragen.

4.3. Entwicklung

Eine weitere Risikostrategie ist die Entwicklung des Prototyps. Er ist ein Bestandteil des Spiralmodells und wird im Rahmen der Arbeit zur Risikovermeidung eingesetzt. Das Spiralmodell definiert für einen Prototyp mehrere Versionen. Die ersten Ideen aus der Anforderungsanalyse des Generators werden im angepassten Zeitplan als 1. Prototyp definiert. Der Geschäftsprozess des Generators wird visuell durch ein Benutzerinterface spezifiziert, das dem Benutzer den ersten Eindruck über

Funktionalitäten vermitteln soll. In Kapitel 5.1.3 wird das Benutzerinterface in Kürze vorgestellt. Die genauen Details können aus dem GUI-Mockup im Pflichtenheft entnommen werden. Der 2. Prototyp enthält die ersten Funktionalitäten, er ist jedoch noch kein vollständiges Produkt. Die Erstellung des 2. Prototyps wird ansatzweise in Kapitel 7 aufgeführt.

5. Anforderungsanalyse

Das Kapitel Anforderungsanalyse definiert in der Regel die erste Phase in der Softwareentwicklung. In dieser Phase werden sowohl funktionale als auch nicht-funktionale Anforderungen an ein System ermittelt. Des Weiteren werden im Zusammenhang mit dem Prototyp Anforderungen hinsichtlich ihrer Relevanz analysiert. Funktionale Anforderungen werden üblicherweise als Anwendungsfälle in UML-Notation modelliert. Der Begriff Modell wird nicht nur im Zusammenhang mit Projektmanagement verwendet. In der Softwareentwicklung bekommt der Begriff im Zusammenhang mit dem Einsatz von Modellierungswerkzeugen noch eine weitere Bedeutung. UML ist ein Beispiel für ein weit verbreitetes Modellierungswerkzeug. Allgemein dient ein Modell der Abstraktion eines Ausschnitts aus der realen Welt um somit die Komplexität des vorliegenden Systems zu reduzieren. Außer der Modellierung von Anwendungsfällen existieren noch weitere Diagrammarten, die entweder eine Systemschicht, ein Verhalten oder einen Zustandsübergang beschreiben. In (Kleuker 2008) werden Notationen in UML, sowie sämtliche Diagrammtypen näher beschrieben. In diesem Kapitel wird das in Kapitel 2 aufgestelltes Grobdesign durch Einsatz von UML weiter spezifiziert.

5.1. Funktionale Anforderungen

5.1.1. Anwendungsfälle

Zunächst stellt sich die Frage, welche Hauptaufgaben im Generator identifiziert werden können. In Kapitel 2 werden erste Thesen zu den Eigenschaften und Funktionalitäten des Generators aufgestellt. Das Grobdesign wird in Kapitel 4.1 als Geschäftsprozess mit einzelnen Komponenten vorgestellt. Sie werden hier noch mal aufgegriffen.

- Datenimport
- Konfiguration
- Simulation

Der Datenimport enthält Funktionen zur Datenkonvertierung und zur Datenspeicherung. Als Komponente des Generators werden Funktionen im Zusammenhang mit Datenhandling als Datenmanagement definiert. Die Komponente Konfiguration ermöglicht dem Benutzer Ereignistypen, Stichproben und Laufzeitparameter zu definieren. Die Definition der Ereignistypen ist für die Verknüpfung auf §21-Daten erforderlich. Die Ereignistypen müssen persistierbar sowie erweiterbar sein und gelöscht werden können. Demnach wird die Verwaltung der Ereignisse als Ereignismanagement-Komponente definiert. Analog ist es beim Stichprobenmanagement mit der Ausnahme, dass die Konfiguration der Stichprobe dateibasiert persistiert wird. Im Zusammenhang mit einer Simulation muss eine Parametrierung vorgenommen werden, die beispielsweise auf die Simulationszeit Einfluss nimmt. Die Simulation selbst muss protokolliert werden. Diese Einstellungen haben primär nichts mit den Ereignissen oder Stichproben zu tun, daher werden sie als Komponente Konfiguration deklariert. Schließlich definiert die Komponente Simulation durch eine Zusammenfassung gewählter Parameter und einen Startknopf den eigentlichen Start einer Simulation. In der Abbildung 20 werden die Anwendungsfälle des Generators zusammenfassend dargestellt.

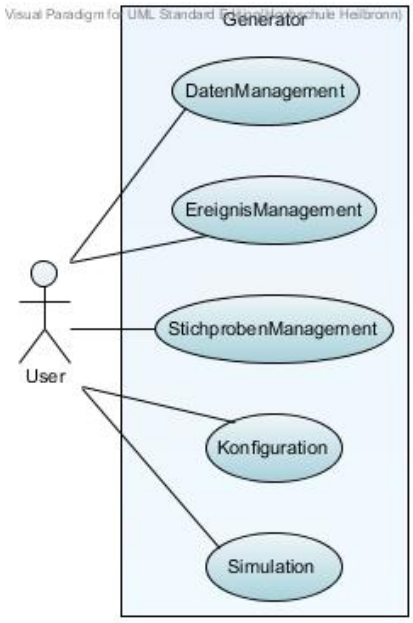


Abbildung 20 Generator mit Komponenten

5.1.1.1. Datenmanagement

Die Komponente Datenmanagement stellt Funktionen zum Konvertieren und Persistieren der Daten zur Verfügung. Zur Konvertierung gehört das Überführen der §21-Daten aus dem CSV-Format nach XML. Zur Simulation müssen die spezifizierten Ereignisse zur Verfügung stehen. Dafür werden sie in einer Datenbank abgelegt. Die Validierung der Daten baut auf mehreren Stufen auf. Weitere Details werden in Kapitel 5.4 näher erläutert. Die Simulation wird mit einigen Parametern belegt, die gesondert im Kapitel 5.1.1.4 betrachtet werden. In der Abbildung 21 wird Datenmanagement in UML dargestellt.

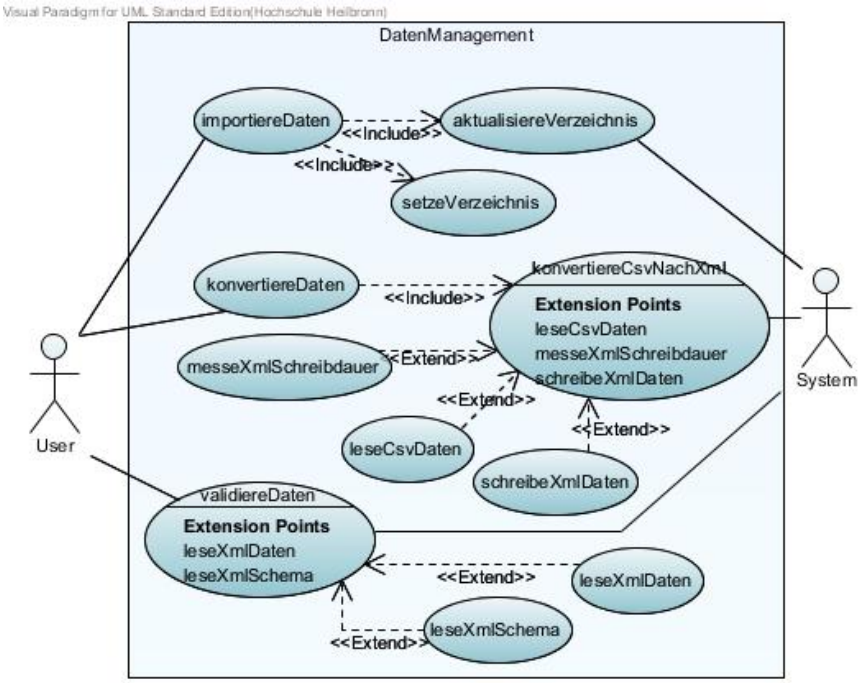


Abbildung 21 Datenmanagement

Im Datenmanagement hat der Benutzer die Möglichkeit zunächst die Daten unter Angabe eines Pfades zu importieren. Es ist an der Stelle wichtig zu unterscheiden, ob die Daten lokal auf dem Client vorliegen und auf den Server geladen werden müssen oder ob sie bereits auf dem Server liegen. Sämtliche Vorbereitungen für eine Simulation werden auf dem Client getroffen. Dazu gehören der Import und das Zusammenstellen von Daten. Im Hinblick auf den Webservice ist es sinnvoll möglichst viel auf einmal über das Internet zu transportieren. Bei den Webservices über SOAP spielt XML beim Austausch von Daten zwischen Client und Server, wie bereits in

Kapitel 3 angesprochen, eine bedeutende Rolle. Das Output des Generators und der nachrichtenbasierte Austausch über XML haben zu der Entscheidung geführt die CSV-Daten bereits beim Import nach XML zu konvertieren. Die Konvertierung erfolgt dateiweise von CSV nach XML. Das bedeutet, dass die Struktur der CSV-Dateien nach der Konvertierung im XML-Format vorliegt. Nach der Konvertierung kann der Benutzer seine XML-Daten gegen ein XML-Schema validieren lassen. Dazu muss ebenfalls die Quelle der XML-Schemata angegeben werden. Des Weiteren wird im Datenmanagement die Schreibdauer einer XML-Datei gemessen. Dazu wird eine Methode definiert, die die Schreibdauer einer XML-Datei ermittelt. Bei der Konvertierung werden die medizinischen Falldaten, also fünf Dateien, als XML-Dateien geschrieben. Statistisch gesehen sind fünf Schreibversuche keine repräsentative Stichprobe. Mit statistischen Mitteln wie Bestimmung des Stichprobenumfangs kann jedoch eine ungefähre Anzahl der notwendigen Schreibversuche aufgestellt werden, um eine repräsentative Stichprobe für eine Schreibdauer zu ermitteln. Der Mittelwert oder besser der Median dieser Stichprobe kann als Referenzwert für die Schreibdauer herangezogen werden. Die Schreibdauer ist ein wichtiger Parameter in der Zeitkomponente und wird unter Anderem zur zeitlichen Steuerung der Simulation verwendet.

5.1.1.2. Ereignismanagement

Der Anwendungsfall Ereignismanagement beschreibt Tätigkeiten des Benutzers zur Verwaltung der Ereignistypen und deren Verknüpfung auf §21-Daten. Die Verwaltung der Ereignistypen kann in der Regel durch CRUD-Methoden erfolgen. Der Benutzer hat mit den CRUD-Methoden die Möglichkeit einen Ereignistyp anzulegen, auszulesen, zu aktualisieren oder zu löschen. Vor dem Ausführen der jeweiligen Methode muss der Ereignistyp im Hintergrund gesucht werden. Die ereignisspezifischen Parameter werden zu einem Ereignistyp gekapselt. Der Ereignistyp wird durch den Namen eines Ereignisses repräsentiert. Für die Verknüpfung der Ereignisse mit einem Behandlungsfall werden die behandlungsfallspezifischen Parameter aus den §21-Daten herangezogen. Als Beispiel können die Parameter, die als DRG-relevant gelten, als Mindestkonfiguration gewählt werden. Weitere Parameter können sich z. Bsp. auf die Zeit/Dauer oder Wahrscheinlichkeit einer Erkrankung beziehen. Mit der Wahl der ereignisspezifischen

und behandlungsfallspezifischen Parameter wird die Verknüpfung eines Ereignistyps mit einem Behandlungsfall zusammengefasst und als Fallereignis abgelegt. In der Abbildung 22 wird die Verknüpfung durch den Anwendungsfall *ordneEreignistypZumBehandlungsfall* dargestellt. Das Ereignismanagement ermöglicht dem Benutzer die Fallereignisse anhand der Ereignistypen und Parametern aus §21-Daten qualitativ zu spezifizieren. Qualitativ bedeutet in dem Fall, es beantwortet die Frage was zu einem Ereignis gehört.

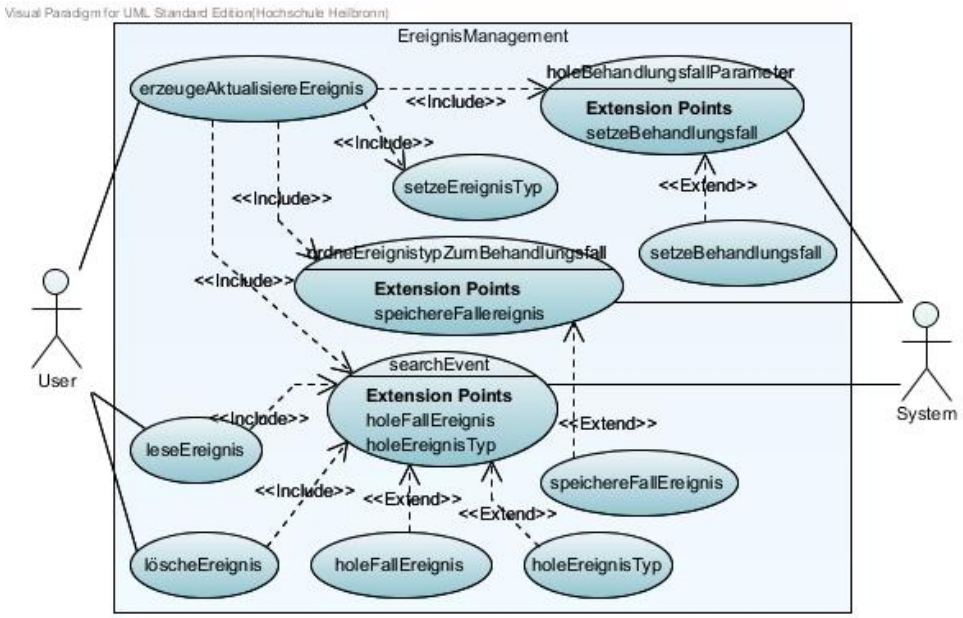


Abbildung 22 Ereignismanagement

5.1.1.3. Stichprobenmanagement

Die Tätigkeiten für das Stichprobenmanagement ähneln denen aus dem Ereignismanagement. Der Unterschied besteht darin, dass im Stichprobenmanagement die Fallereignisse durch die Wahl der behandlungsfallspezifischen Parameter zu ereignisbasierten Nachrichtenprofilen gefiltert werden. Die Parameter werden für die Filterung, wie im Ereignismanagement, aus §21-Daten bezogen. Außerdem werden die Fallereignisse durch einen Zeitrahmen beschränkt. Ein weiterer Aspekt im Stichprobenmanagement ist das Abspeichern der Stichprobenkonfiguration. Das bedeutet, dass der Benutzer seine

zusammengestellte Stichprobe als Konfigurationsdatei lokal auf dem Datenträger ablegen und bei Bedarf wieder einlesen kann.

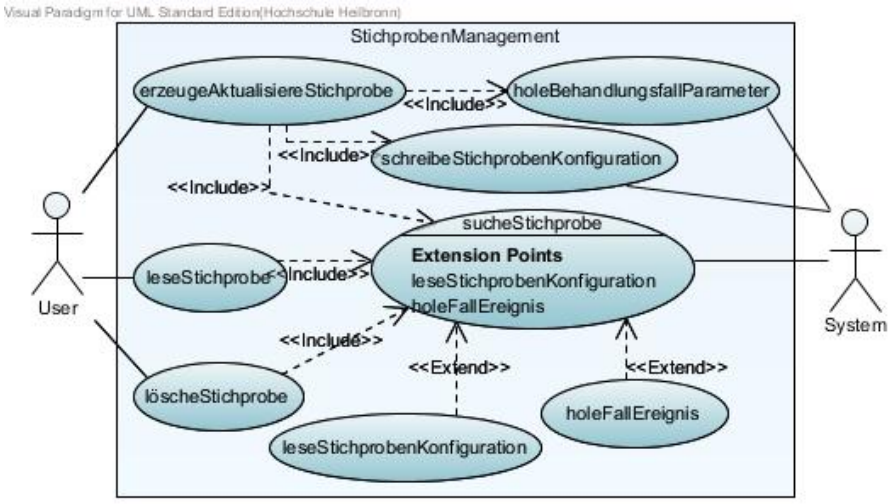


Abbildung 23 Stichprobenmanagement

In der Abbildung 23 wird das Stichprobenmanagement als Anwendungsfall abgebildet. Durch Stichprobendefinitionen werden die Fallereignisse quantitativ eingegrenzt. In dem Fall können die Fallereignisse durch Konstellationen der Parameter, wie beispielsweise das Geschlecht und Alter oder das Geschlecht und die Diagnose oder aber auch das Alter und die Diagnose, in der Menge eingegrenzt werden.

5.1.1.4. Zeitkomponente

Die Zeitkomponente ist für das Abarbeiten der Fallereignisse aus einer Stichprobe zuständig. Die Ereignisse werden im Zusammenhang mit der Systemzeit in eine relative zeitliche Abfolge gebracht. Die Zeitkomponente steht mit allen aufgeführten Komponenten des Generators in Beziehung. Aus dem Datenmanagement wird die Schreibdauer herangezogen. Aus dem Ereignismanagement werden der Zeitpunkt, der Vorgänger und der Abstand eines Fallereignisses sowie deren Anzahl zusammengestellt. Das Stichprobenmanagement liefert den Zeitrahmen für die

Stichprobe. Diese Parameter liefern die Grundlage für die Abbildung der Fallereignisse auf einer Zeitachse.

Zunächst ist es sinnvoll festzustellen, ob im Bezug auf die Systemzeit ein Schaltjahr vorliegt, denn ein Schaltjahr als Zeitspanne betrachtet, ist länger als sonst. Ebenfalls muss die Zeitumstellung berücksichtigt werden. Die Unterscheidung hat einen Einfluss darauf, wenn die Ereignisabstände auf der Zeitachse abgebildet werden und die Ereignisse als XML-Dateien auf die Festplatte geschrieben werden. Die Anzahl der XML-Dateien, die in diesem Zeitraum geschrieben werden kann, wird durch das Verhältnis der Periodendauer zu Schreibdauer gebildet. Diese Beziehung wird für die Definition des Zeitraffungsfaktors zugrunde gelegt.

Der Zeitrahmen aus dem Stichprobenmanagement wird durch den Benutzer auf zwei Wegen beschränkt. Zum einen wird die Stichprobe zeitlich eingegrenzt und zum anderen wird die Anzahl der Ereignisse durch Wahl der Parameter aus §21-Daten reduziert. Das Ergebnis ist eine Stichprobenperiode mit Fallereignissen, die durch den Zeitpunkt, Vorgänger und Abstand auf einer Zeitachse abgebildet werden können. Für die Abbildung der Ereignisse wird jeder definierte Ereignistyp mit einem Nullpunkt versehen.

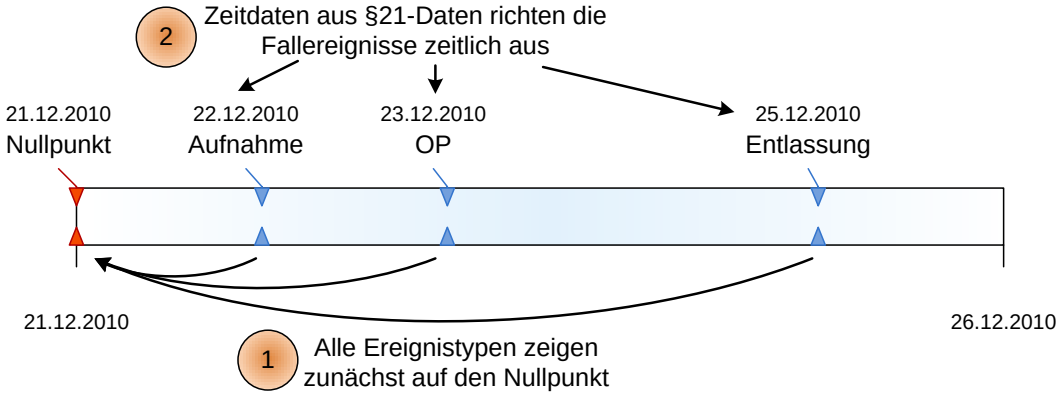


Abbildung 24 Vom Ereignistyp zum Fallereignis

Die Abbildung 24 beschreibt den Sachverhalt, dass alle definierten Ereignistypen zunächst auf denselben Nullpunkt zeigen. Durch Auslesen der Zeitdaten aus den §21-Daten werden die Fallereignisse zeitlich ausgerichtet. Mit der Ausrichtung wird der korrekte syntaktische Zusammenhang der Ereignisse hergestellt. Für eine Simulation müssen die Fallereignisse jedoch in einem Bezug zueinander gebracht werden. Dazu wird eine Ereigniskette aus Abbildung 24 zugrunde gelegt. Der

Nullpunkt definiert den Ursprung und hat keinen Vorgänger. Das Ereignis Aufnahme kann logischerweise nur den Nullpunkt als Vorgänger haben. Das Ereignis OP kann eine weitere OP oder die Aufnahme als Vorgänger haben. Die Entlassung kann entweder eine OP oder eine Aufnahme als Vorgänger haben. Die Betrachtung von Abstand und Vorgänger der Ereignisse lässt auf deren Semantik schließen. Die Betrachtung der Vorgänger und der Abstände ist nur möglich, wenn die Ereigniskette einem Patienten zugeordnet werden kann. Bei den §21-Daten handelt es sich jedoch um anonymisierte Falldaten, das bedeutet, dass die Ereignisse ohne konkreten Bezug zu einem Patienten stehen. Zur Patientenidentifikation bedarf es mehr an Informationen, wie z. Bsp. die Patientennummer. In den §21-Testdaten von 2006 wird zwar die Patientennummer mitgeführt. In der Praxis wird dieses Datenfeld jedoch nicht gefüllt. In dem Fall wird eine Ausnahme definiert und statt der Patientennummer das krankenhausinterne Kennzeichen zur Identifikation verwendet. Die Fallereignisse werden anhand des krankenhausinternen Kennzeichens zu einem Behandlungsfall zugeordnet. In der Abbildung 25 wird die Stichprobenperiode für einen Behandlungsfall beispielhaft veranschaulicht. Die Fallereignisse haben einen Vorgänger und durch Differenz lässt sich der Abstand ermitteln.

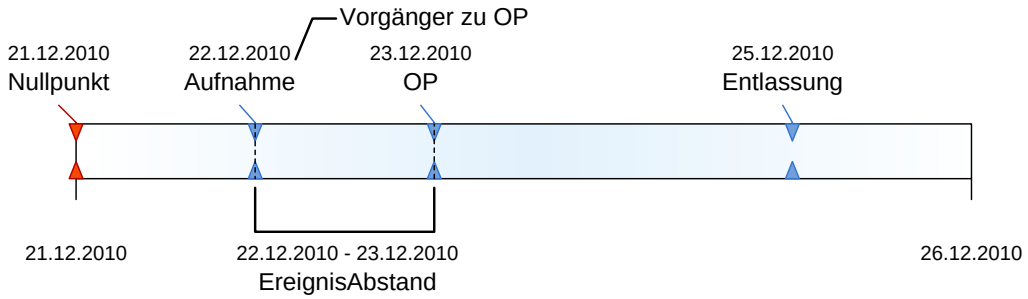


Abbildung 25 Zeitachse mit Ereignissen

Als Nächstes wird die Schreibdauer für ein XML-Ereignis betrachtet. Zu der Zeit liegen für das Schreiben einer XML-Datei keine Erfahrungswerte vor. Im Hinblick auf den Einsatz von JAXB werden aus (JAXB zur XML-Verarbeitung) Praxiswerte für das Parsen von XML-Dateien herangezogen. Die folgende Tabelle gibt einen Überblick.

Größe der XML-Datei	Anzahl der XML-Elemente	JAXB ohne Stax ohne Validierung
1KB	10	0,01s
8MB	100.000	0,5s
76MB	1.000.000	5s

Tabelle 3 JAXB-Parsing Schätzwerte

Die Werte aus der Tabelle 3 beschreiben das Einlesen von XML-Dateien, die in der Größe sowie in der Anzahl von XML-Elementen variieren. Unter der Annahme, dass die Festplatte gleich schnell schreibt wie sie liest und dass beim Schreiben nicht validiert werden muss, wird aus Tabelle 3 als Schätzwert die Schreibdauer für eine XML-Datei mit 0,5 Sekunden angenommen. Aus der Beziehung der Schreibdauer (SD) und der Stichprobenperiode (SPP) ergibt sich die maximale Anzahl an XML-Dateien (XML_{max}) wie folgt.

$$XML_{max} = \frac{SPP}{SD}$$

Formel 1 Maximale Anzahl der XML-Dateien

Die aus Formel 1 ermittelte Anzahl für XML-Dateien berücksichtigt lediglich den Festplattenzugriff. Es ist bekannt, dass der Festplattenzugriff sich auf Millisekunden beläuft, wodurch die zu berücksichtigende Zugriffslücke um den Faktor 10^5 zwischen Arbeitsspeicher und Festplatte entsteht. Zugriffszeiten sonstiger Hardware, wie der CPU oder des Arbeitsspeichers sind in der Hinsicht vernachlässigbar, weil sie im Nanosekundenbereich liegen.

Die Simulation soll im Hinblick auf die voraussichtliche Dauer durch Zeitraffung beeinflusst werden können. Die voraussichtliche Dauer hängt neben der Schreibdauer auch von der Anzahl der Fallereignisse ab. Die Anzahl der Ereignisse hängt wiederum davon ab, was der Benutzer als Ereignistyp definiert, je mehr Ereignistypen, desto größer ist die Ereignismenge. Ein Ereignistyp wird durch den Benutzer anhand der Parameter aus den §21-Daten spezifiziert. Prinzipiell sind nur die Parameter relevant, die auf der Zeitachse abgebildet werden können. Dieser Aspekt wird in Kapitel 5.4 noch näher präzisiert. Unabhängig davon stehen die

Anzahl der Ereignisse (AE) und die Schreibdauer (SD) für eine XML-Datei in folgender Beziehung.

$$GSD = AE * SD$$

Formel 2 Gesamtschreibdauer der XML-Dateien

Die Gesamtschreibdauer für beispielsweise drei Ereignisse aus Formel 2 ergibt 1,5 Sekunden. Sie beschreibt zwar den technischen Aspekt – nämlich wie lange es dauert die Ereignisse als XML-Dateien zu schreiben. Daraus ist jedoch nicht ersichtlich, in welchem Bezug die Ereignisse zueinander stehen. In der Abbildung 25 wurde bereits angedeutet, dass Fallereignisse durch das krankenhausinterne Kennzeichen in eine zeitliche Reihenfolge gebracht werden können. In Abbildung 26 wird dieser Sachverhalt unter Berücksichtigung der Schreibdauer wie folgt dargestellt.

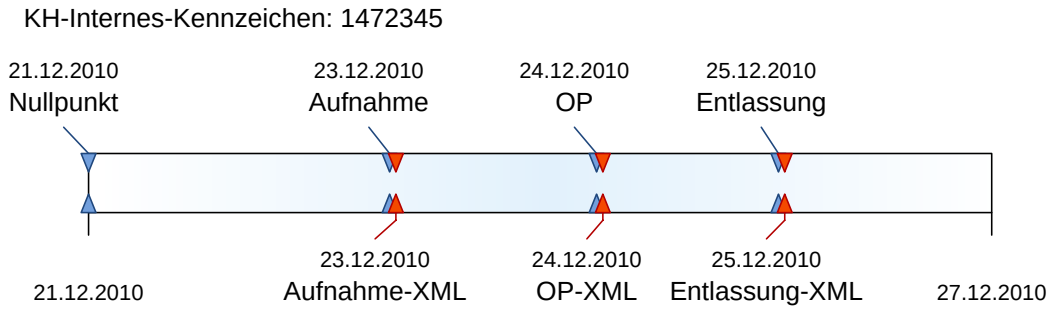


Abbildung 26 Beziehung der Parameter Anzahl der Ereignisse, Abstand und Schreibdauer

Die Zeitpunkte für das Schreiben der Fallereignisse sind wegen der Schreibdauer jeweils um 0,5 Sekunden versetzt. Die Fallereignisse gehören einem Behandlungsfall, das ermöglicht die Fallereignisse in Bezug zueinander zu setzen und den Abstand zu ermitteln. Es wird eine Größe benötigt, die zum Einen die Anzahl der Fallereignisse liefert und zum Anderen zeitlich abgebildet werden kann. Jedes Fallereignis hat einen Abstand, der zeitlich abgebildet werden kann. Die Anzahl der Abstände liefert die Anzahl der Fallereignisse. Somit lässt sich die voraussichtliche Dauer (VD) einer Stichprobenperiode wie folgt definieren.

$$VD = \sum_{i=1}^{i=AE} (A_i + SD)$$

Formel 3 Voraussichtliche Dauer einer Stichprobenperiode

So gesehen ist es nicht möglich die voraussichtliche Dauer von außen zu beeinflussen, weil die Größen durch das System vorgegeben werden. Durch das Hinzufügen einer weiteren Größe, nämlich des Zeitraffungsfaktors, im Folgenden als Simulationsfaktor (SF) bezeichnet, wird die Steuerung der Simulation durch Benutzer ermöglicht. Die Beziehung zwischen VD, Simulationsfaktor (SF), Abstand (A_E) und Schreibdauer (SD) wird wie folgt definiert.

$$VD = \sum_{i=1}^{i=AE} (SF * A_E + SD) \; mit \; \{SF \in \mathbb{Q} : \; 0 < SF \leq 1\}$$

Formel 4 Voraussichtliche Dauer mit Zeitraffung

Die voraussichtliche Dauer aus Formel 4 berücksichtigt den zeitlichen Abstand der Ereignisse und die Schreibdauer. Durch Multiplikation des Simulationsfaktors mit dem Abstand eines Ereignisses wird die Zeitraffung für den Abstand beeinflusst. Das anschließende Schreiben der XML-Datei wird durch Addition der Schreibdauer ergänzt. Aus der Anzahl der Abstände innerhalb der Stichprobenperiode lässt sich Anzahl der Ereignisse ableiten. Sie wird in der Summe als Grenzwert verwendet. Der Simulationsfaktor aus Formel 5 schließt den Wert null als ganze Zahl aus, weil die Schreibdauer allein keinen Bezug zur Reihenfolge der Ereignisse hat. Der untere Grenzwert ≤ 1 beschreibt den realen Simulationsablauf. Zur Bestimmung des oberen Grenzwertes wird das Verhältnis aus Anzahl der vorliegenden Ereignisse und Anzahl der maximal möglichen Ereignisse in XML herangezogen. Die Anzahl der maximal möglichen Ereignisse in XML wird durch die Formel 1 definiert. Somit wird folgende Beziehung für den oberen Grenzwert definiert.

$$SF_{max} = \frac{AE}{XML_{max}}$$

Formel 5 Grenwertbestimmung für Zeitraffung

Die voraussichtliche Dauer aus Formel 4 und die Grenzwerte bilden den Zeitraffungsfaktor ab, der durch den Benutzer an der Oberfläche gesteuert werden kann. Die Steuerung wird im Anwendungsfall Konfiguration beschrieben.

5.1.1.5. Konfiguration

Im Anwendungsfall Konfiguration hat der Benutzer die Möglichkeit, den Zeitraffungsfaktor aus der Zeitkomponente zu setzen. Der Zeitraffungsfaktor variiert im Rahmen der ermittelten Grenzen, die systembedingt sind. In der Abbildung 27 wird mit dem *errechneVoraussichtlicheSimulationsdauer*-Anwendungsfall der gesetzte Zeitraffungsfaktor durch den Benutzer aktualisiert. Vorher ermittelt das System die maximale Anzahl der XML-Ereignisse und die Schreibdauer. Der Zusammenhang der Parameter wurde bereits in Kapitel 5.1.1.4 beschrieben. Desweiteren hat der Benutzer die Möglichkeit den Simulationsvorgang zu protokollieren. Zur Übersicht werden Anzahl der zu generierenden XML-Ereignisse und Stichprobenkonfiguration mit gesetzten Parametern dargestellt.

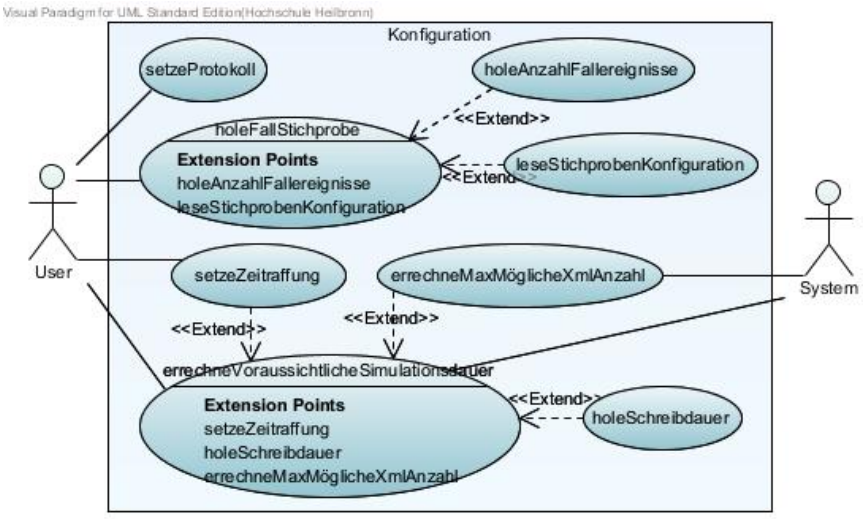


Abbildung 27 Konfiguration

Die Einstellungen werden an den Anwendungsfall Simulation weitergereicht, in dem die Simulation durch den Benutzer gestartet werden kann.

5.1.1.6. Simulation

Der Anwendungsfall Simulation, siehe Abbildung 28, spezifiziert das Starten oder Abbrechen eines Simulationsvorganges. Alle Informationen, die zuvor gesammelt werden, werden hier zusammengefasst dargestellt.

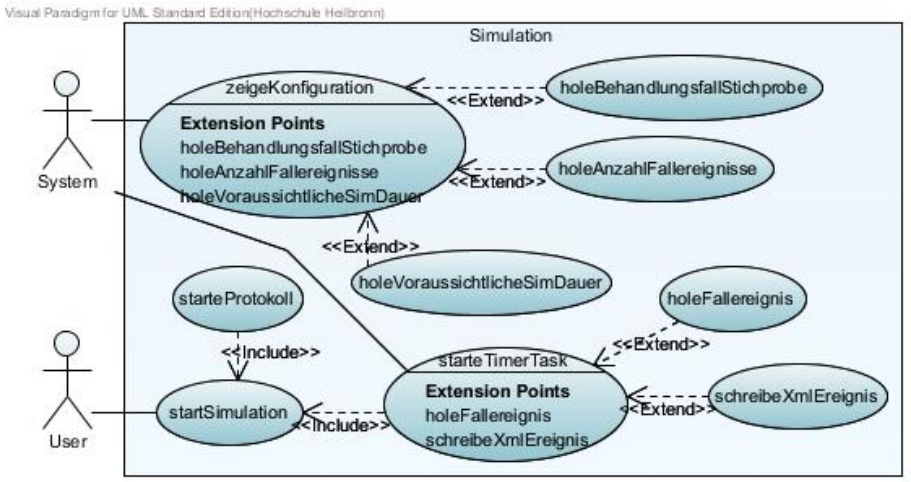


Abbildung 28 Simulation

Mit dem Start des Simulationsvorganges wird durch das System ein Timertask gestartet, der zum Ereigniszeitpunkt ein Ereignis als XML-Datei schreibt.

5.1.2.Systemverhalten

Die Anwendungsfälle ermitteln die Funktionalitäten des Generators. Aus den Anwendungsfällen ist jedoch nicht ersichtlich, wie der Generator im Detail auf Aktionen des Benutzers reagiert. Das Verhalten eines Systems lässt sich mit einer weiteren UML-Diagrammart – Aktivitätsdiagramme – beschreiben. Sie beschreiben schrittweise die Teilprozesse, die als ein Anwendungsfall zusammengefasst werden. Im Folgenden werden für die Beschreibung der Aktivitäten die Anwendungsfälle aus der Anforderungsanalyse zugrunde gelegt.

5.1.2.1. Datenmanagement

Das Datenmanagement spezifiziert Methoden zum *Importieren*, *Konvertieren* und *Validieren* von §21-Daten. Die Methode *Konvertieren* impliziert das Persistieren der konvertierten Daten. Der Import der §21-Daten beruht auf dem Einlesen der – in Kapitel 2.3 definierten – CSV-Dateien. An dieser Stelle gibt es zwei Möglichkeiten zur Weiterverarbeitung – siehe in Kapitel 4.2.1.1. Der gewählte Ansatz beschreibt als nächsten Schritt die Konvertierung der §21-Daten von CSV nach XML. Die entsprechende Methode liest aus dem angegebenen Verzeichnis die CSV-Dateien und schreibt sie als XML-Dateien. Die konvertierten XML-Daten können gegen ein XML-Schema validiert werden[8]. Während der Konvertierung – Einlesen der CSV-Dateien und Schreiben als XML-Dateien – wird für jede geschriebene XML-Datei die Schreibdauer gemessen. Der Zweck der Schreibdauer wird in 5.1.4 Zeitkomponente beschrieben. Für die Schritte Import, Konvertieren und Validieren muss der Benutzer jeweils ein Verzeichnis angeben. Bei erfolgreicher Validierung liegen die §21KHEntG-Daten im XML-Format vor. Falls die Validierung nicht erfolgreich verläuft, so wird der Benutzer benachrichtigt. Das Ergebnis aus dem Datenmanagement wird durch die §21-Datenmenge im XML-Format und die Schreibdauer für eine XML-Datei definiert. In der Abbildung 29 wird der Ablauf des Datenimportes im Aktivitätsdiagramm veranschaulicht.

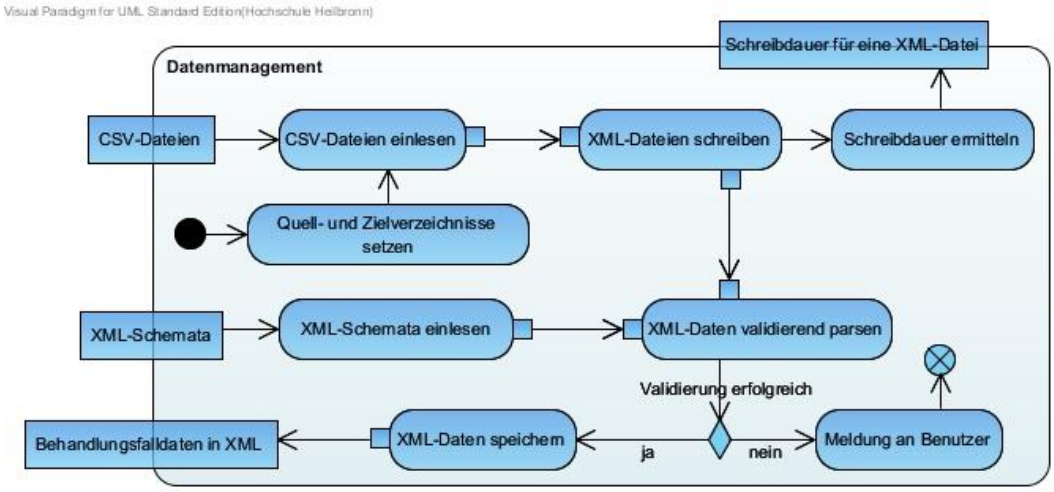


Abbildung 29 Aktivitäten im Datenmanagement

[8] Aus Übersichtlichkeitsgründen wurde in der Abbildung auf die Darstellung des alternativen Imports ohne anschließende Validierung verzichtet.

5.1.2.2. Ereignismanagement

Im Ereignismanagement werden zum Einen Ereignistypen und zum Anderen Fallereignisse definiert. Die Ereignistypen bestehen aus ereignisspezifischen Parametern, die zusammen mit behandlungsfallspezifischen Parametern ein Fallereignis definieren. Sobald die Daten aus dem Datenmanagement vorliegen, ist es für den Benutzer möglich ein Fallereignis anzulegen. Wenn der definierte Ereignistyp nicht vorhanden ist, wird er samt seiner Parameter gespeichert. Im Fall, dass der Ereignistyp bereits vorliegt, wird der Benutzer benachrichtigt. Zur Vereinfachung wird diese Abfrage in der Abbildung 30 nicht abgebildet. Nach der Speicherung des Ereignistyps stehen die ereignisspezifischen Parameter für die Verknüpfung mit den §21-Daten zur Verfügung. Für die Verknüpfung der Ereignistypen mit einem Fallereignis werden die behandlungsfallspezifischen Parameter nach Relevanz ermittelt. Der Gedanke ist bereits im Anwendungsfall Stichprobenmanagement erwähnt worden, dass möglichst die Parameter zur Verfügung gestellt werden, die für zeitgesteuerte oder wahrscheinlichkeitsbasierte Ereignisse relevant sind. Nachdem ereignisspezifische und behandlungsfallspezifische Parameter durch den Benutzer gesetzt sind, wird eine Plausibilitätsprüfung durchgeführt. Anschließend wird das Fallereignis gespeichert. Beim Speichern wird ebenfalls überprüft, ob das Fallereignis bereits existiert. Die Abbildung 30 beschreibt den Ablauf.

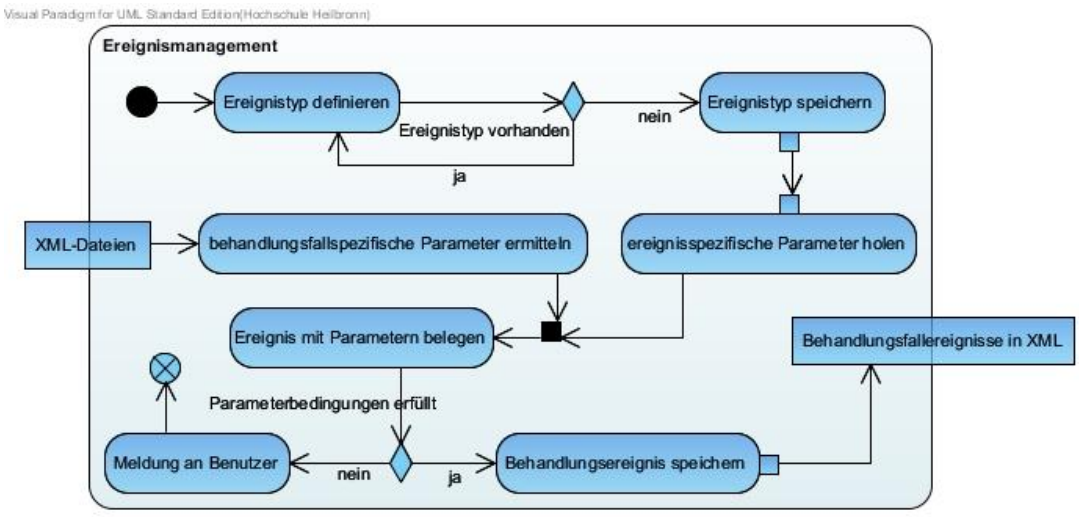


Abbildung 30 Aktivitäten im Ereignismanagement

5.1.2.3. Stichprobenmanagement

Im Stichprobenmanagement, siehe Abbildung 31, wird eine Stichprobenkonfiguration für die Fallereignisse aus dem Ereignismanagement angelegt. Die Stichprobenkonfiguration enthält wie der Ereignistyp ihre spezifischen Parameter. Hauptsächlich geht es um die Beschränkung des Zeitrahmens. Zu den stichprobenspezifischen Parametern müssen wiederum die behandlungsspezifischen Parameter geladen werden. Sie werden durch den Benutzer gewählt, dennoch wird eine Vorauswahl der Parameter getroffen, um die Fallereignisse nach Eigenschaften zu filtern. Eigenschaften können beispielsweise Geschlecht, Diagnose oder Alter sein. Wenn alle Parameter für eine Stichprobe vorliegen, wird diese Auswahl auf die Fallereignisse abgebildet. Anschließend werden Plausibilitätsprüfungen durchgeführt. Im Fehlerfall wird der Benutzer benachrichtigt. Als Ergebnis aus dem Stichprobenmanagement liegen ereignisbasierte Nachrichtenprofile vor, die mit Hilfe der Zeitkomponente verarbeitet werden.

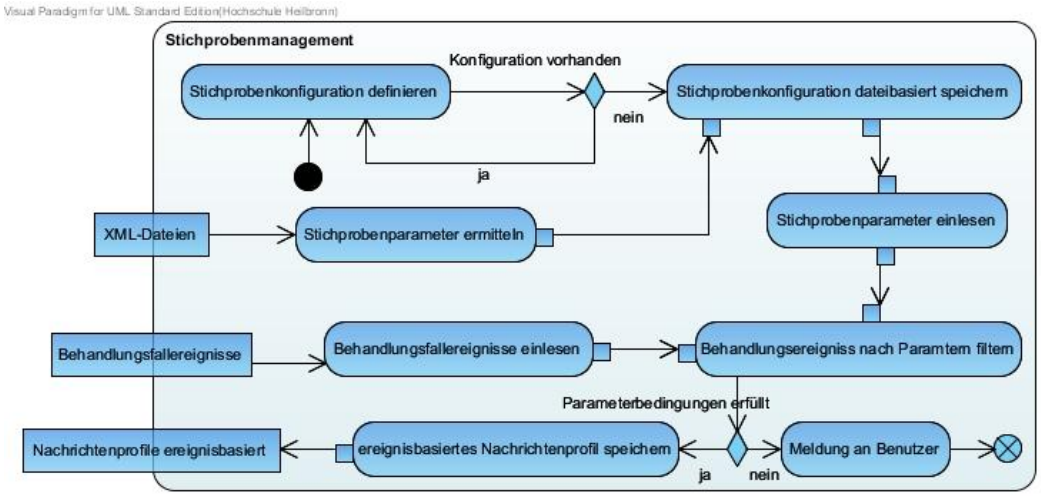


Abbildung 31 Aktivitäten im Stichprobenmanagement

5.1.2.4. Konfiguration

In der Konfiguration werden folgende Aktionen ausgeführt. Der Benutzer setzt den Zeitraffungsfaktor. Dafür werden weitere Parameter in der Abbildung 32 benötigt, die zuvor ermittelt werden. Das Aktualisieren des Timers im Hintergrund muss der

Benutzer bestätigen. Mit dem aktuellen Zeitraffungsfaktor wird die voraussichtliche Dauer für eine Simulation ermittelt und dem Benutzer wieder als Information angezeigt. Weiterhin hat der Benutzer die Möglichkeit eine Simulation zu protokollieren, indem er die Logging-Funktion aktiviert. Als Ergebnis werden die Nachrichtenprofile und der Zeitraffungsfaktor an die nächste Komponente – Simulation weitergereicht.

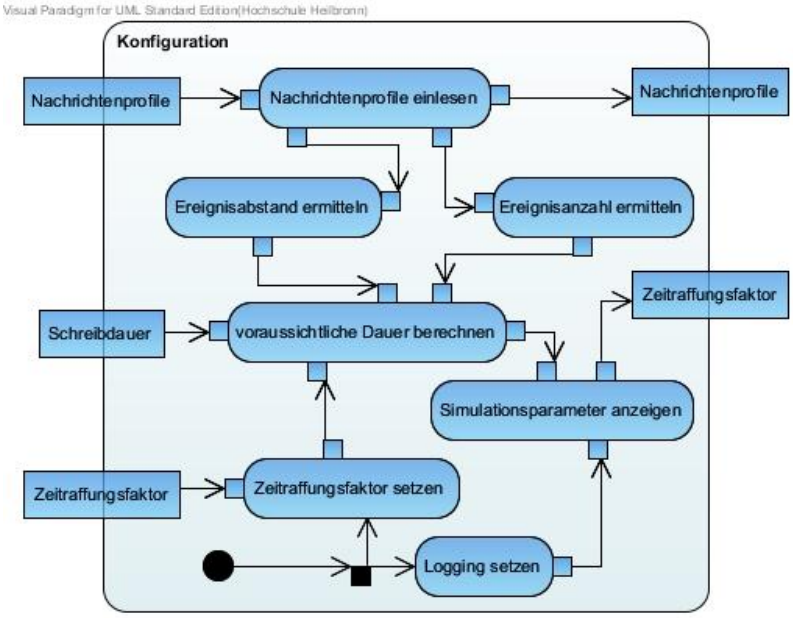


Abbildung 32 Aktivitäten in der Konfguration

5.1.2.5. Simulation

Sobald alle notwendigen Daten vorliegen ist es für den Benutzer möglich die Simulation per Startknopf zu starten. Die Abbildung 33 veranschaulicht die Aktivitäten der Simulation.

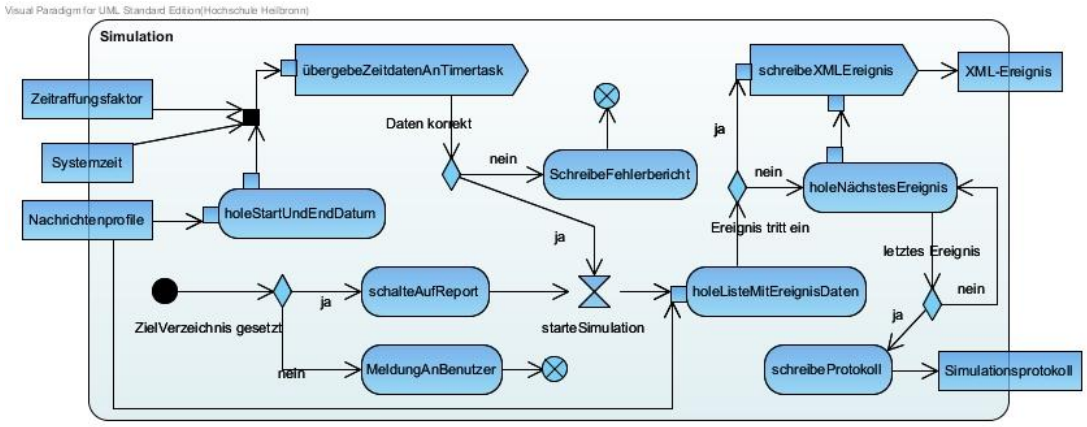


Abbildung 33 Aktivitäten in der Simulation

Zu den Daten gehören die Nachrichtenprofile, der Zeitraffungsfaktor und die Systemzeit. Die Bezeichnung für Nachrichtenprofile wird an der Stelle für gefilterte Fallereignisse verwendet. Wenn eines davon nicht vorliegt, wird die Simulation durch eine Fehlermeldung abgebrochen. Beim Start werden die Nachrichtenprofile der Zeitkomponente übergeben. Die Nachrichtenprofile werden in eine Reihenfolge gebracht und von einem Timertask verarbeitet. Die Reihenfolge wird anhand des Vorgängers und des Abstandes des jeweiligen Fallereignisses ermittelt. Für jedes Fallereignis wird ein Timertask erzeugt, der im Abstand des Fallereignisses eine XML-Datei auf Festplatte schreibt. Bei erfolgreicher Simulation wird eine Protokolldatei geschrieben, die alle vor der Simulation angezeigten Informationen als eine Datei schreibt.

5.1.3.Benutzerinterface

Der Generator wird über ein Benutzerinterface konfiguriert, welches über eine URL im Browser zur Verfügung steht. Die Funktionalitäten werden auf der Seite durch jeweils einen Verweis repräsentiert. Die Elemente der Darstellung basieren auf GUI-Elemente in Java-Entwicklung. Hinsichtlich der Umsetzung der Oberfläche des Generators gibt es im Zusammenhang mit Spring mehrere Möglichkeiten. In Kapitel 3 wurde die Technologie mit JSP und JSTL in Erwägung gezogen. Im Spiralmodell wird

mit dem ersten Prototyp häufig ein GUI-Mockup[9] erstellt, um die Funktionalitäten auf der Benutzeroberfläche dem Benutzer zu vermitteln. Für weitere Details zum Generator-GUI-Mockup wird auf das Pflichtenheft verwiesen.

5.2. Nichtfunktionale Anforderungen

Zu den nichtfunktionalen Anforderungen gehören in der Regel Qualitätsaspekte eines Systems. Im Generator-Projekt wurden folgende Qualitätsaspekte als relevant ermittelt.

- Funktionalität
- Interoperabilität
- Benutzbarkeit
- Performanz
- Wartbarkeit und Erweiterbarkeit
- Sicherheit
- Rechtliche Anforderungen

Die Prototypentwicklung nach dem Spiralmodell liefert an sich kein Endprodukt. Insofern können hinsichtlich der Umsetzung nicht alle Qualitätsaspekte berücksichtigt werden. Die Konzeption sieht jedoch vor, hinsichtlich der Fortsetzung des Generator-Projektes, Ansätze zur Erfüllung der aufgezählten Qualitätsaspekte zu definieren.

Die Funktionalitäten werden durch Analyse der Anwendungsfälle konzeptuell zusammengetragen. Aufgrund der Änderungen im Zeitplan werden in Kapitel 5.3 konkrete Funktionalitäten definiert, die in Kapitel 7 umgesetzt werden.

Die Interoperabilität ist ein wichtiges Kriterium im Gesundheitswesen. Dieser Aspekt wird im Virtuellen Gesundheitssystem ebenfalls eine wichtige Rolle spielen, wenn später weitere Akteure hinzukommen. Im Generator-Projekt wird die Interoperabilität durch den Ansatz verfolgt, die Simulation der Fallereignisse als Webservice zur Verfügung zu stellen.

Die Erstellung von GUI-Mockups ermöglichen zum einen Schlüsse auf Benutzbarkeit und zum anderen werden die Funktionalitäten veranschaulicht, die das System

[9] GUI-Mockup: Designentwurf der Oberfläche bzw. des Benutzerinterface.

erfüllen soll. Das Benutzerinterface wird mit dem ersten Prototyp im Pflichtenheft veranschaulicht. Des Weiteren werden aus dem GUI-Mockup die Funktionalitäten des Generators ersichtlich.

Aufgrund der §21-Datenmenge sollte der Generator bei der Simulation sowie bei Anfrage- und Antwortzeiten performant sein. Dieser Aspekt hängt zum großen Teil von der eingesetzten Hardware ab. Hinsichtlich der Verarbeitung der Daten werden die Zeiten des Systems gemessen, um mögliche Schlüsse über die Geschwindigkeit des Generators ziehen zu können.

Der Einsatz von Spring ermöglicht mit *Application Context* und *Dependcy Injection* lose Kopplung der einzelnen Komponenten. Durch Einsatz weiterer Entwurfsmuster wird der Aspekt Wartbarkeit und Erweiterbarkeit gewährleistet.

Der Aspekt Sicherheit gehört üblicherweise zu den Anforderungen, jedoch wird dieser Aspekt im Generator-Projekt irrelevant. Zum einen sind die §21-Falldaten anonymisiert und zum anderen wird in der Entwicklung des Generators keine Benutzerverwaltung gefordert. Die Zusammenstellung der Stichproben sind einzelne, benutzereigene Daten, die der Benutzer selbst verwaltet. An der Stelle wird der Aspekt Sicherheit auf den Administrator des Rechensystems übertragen.

Das §21KHEntg-Gesetz regelt die Übermittlung der Falldaten. Die rechtlichen Anforderungen regeln somit den Input des Generators und sollten aufgrund möglicher Gesetzesänderungen im weiteren Verlauf des Generator-Projektes berücksichtigt werden.

5.3. Anforderungen an den Prototyp

Das Ergebnis der Risikoanalyse wird in diesem Abschnitt auf die Prototypentwicklung übertragen. Neue Erfahrungen mit den Technologien haben im Zeitplan zu Verzögerungen geführt. Infolgedessen werden die Anforderungen an den Generator neu definiert, die das sogenannte K.O.-Kriterium[10] erfüllen sollen. Sie werden in die Prototypentwicklung einfließen.

[10] Das K.O-Kriterium (Pfetzing & Rohde 2009 S.214) definiert Anforderungen ohne die kein Betrieb des Generators möglich wäre.

Das Ziel ist die Simulation der ereignisbasierten Nachrichtenprofile. Dieser Geschäftsprozess besteht aus mehreren Einzelprozessen, die im Laufe der Anforderungsanalyse in Komponenten spezifiziert werden. Die Voraussetzung für den Betrieb wird durch die Komponenten Datenmanagement und Simulation spezifiziert. Aus den beiden Komponenten werden für die Prototypentwicklung folgende Funktionalitäten extrahiert.

- Einlesen und konvertieren der Falldaten nach XML.
- Bilden der Behandlungsfälle, die zur weiteren Verarbeitung in eine objektorientierte Datenbank abgelegt werden.
- Anlegen der Ereignistypen. Als Mindestanforderung gelten Aufnahme-, OP- und Entlassungsereignisse.
- Verknüpfen der Ereignistypen und Behandlungsfälle zu Fallereignissen.
- Simulieren der Fallereignisse in der Zeitkomponente und schreiben dieser als XML-Datei.

Diese Anforderungen bilden die Grundfunktionalitäten und können später bei der Wiederaufnahme des Projektes hinsichtlich der Stichprobenverwaltung weiter ausgebaut werden.

5.4. Anforderungen an §21-Daten

In Kapitel 2.3 wurde die Datenstruktur der §21-Falldaten vorgestellt und hinsichtlich relevanter Datenfelder für die Ereignistypen Aufnahme, OP und Entlassung analysiert, die zur Konstruktion der Fallereignisse eingesetzt werden. Des Weiteren wurden Informationen zur Validierung der Daten gesammelt, die bei der Testspezifikation eingesetzt werden.

Den Kern für einen Behandlungsfall definiert die Datei Fall. Die Bedeutung einzelner Datenfelder kann in der Vereinbarung der *InEK,* siehe Externe Dokumente, nachgesehen werden. Für den Prototyp und die Konstruktion der Ereignistypen Aufnahme, OP und Entlassung werden folgende Datenfelder in Betracht gezogen.

- Das krankenhausinterne Kennzeichen dient der Identifikation des Behandlungsfalls und wird neben der Datei Fall auch in den Dateien ICD, FAB, OPS und ENTGELTE eingesetzt.
- Die Patientennummer dient der Identifikation des Patienten. Allerdings wird sie in der Praxis aus Gründen der Anonymität nicht mitgeliefert.
- Geburtsjahr-, Monat- und Tag sind einzelne Datenfelder und definieren das Geburtsdatum. In den Testdaten fehlen allerdings mehrfach Angaben zum Tag und Monat, so dass lediglich das Datenfeld Geburtsjahr zur Verarbeitung herangezogen wird. Im Fall, dass ein Kind unter einem Jahr aufgenommen wird, muss der Monat angegeben werden.
- Das Geschlecht wird bei der Stichprobendefinition relevant, wenn es darum geht bestimmte Fälle nach dem Geschlecht zu filtern.
- Das Aufnahme- und Entlassungsdatum spezifizieren das Datum, an dem ein Fall aufgenommen oder entlassen wird. Sie werden für die Fallereignisse Aufnahme und Entlassung eingesetzt. Bei dem Entlassungsdatum gilt zu unterscheiden, ob der Fall entlassen oder verlegt wird.

Die Datei Fall definiert eine 1:n-Beziehung zu den Dateien ICD, FAB, OPS und Entgelte. Die Datei ICD enthält Informationen über Diagnosen zu einem Behandlungsfall. Hier werden folgende Datenfelder verwendet.

- Der ICD-Code beschreibt die Diagnose des Behandlungsfalls.
- Die Diagnoseart unterscheidet zwischen Haupt- und Nebendiagnose.
- Die ICD-Version gibt Auskunft über das Jahr, aus dem die ICD-Klassifikation stammt.

Für den Ereignistyp OP werden aus der Datei OPS folgende Datenfelder herangezogen.

- Der OPS-Code beschreibt die medizinische Tätigkeit am Patienten. Alle Tätigkeiten, die mit einer OP zusammenhängen, werden im fünften Kapitel des OPS-Katalogs geführt. Für die Datenfelder bedeutet es, dass nur die OPS-Codes mit der Ziffer 5 an der ersten Stelle relevant sind.
- Das OPS-Datum liefert das Datum der OP und wird für Fallereignisse vom Typ OP eingesetzt.
- Die OPS-Version spezifiziert das Jahr des OPS-Katalogs.

Des Weiteren ist Datei FAB hinsichtlich der Konstruktion der Ereignistypen Aufnahme und Entlassung relevant. Sie registriert den Wechsel der Fachabteilungen bei einem Behandlungsfall. Dazu werden folgende Datenfelder eingesetzt.

- Das Datenfeld FAB kennzeichnet die Fachabteilung.
- Die Felder FAB-Aufnahme- und Entlassungsdatum geben Auskunft, wann ein Behandlungsfall in eine Fachabteilung aufgenommen oder entlassen wird. Im Zusammenhang mit den Aufnahme- und Entlassungsdaten der Datei Fall liegen die Daten der FAB-Datei zeitlich gesehen innerhalb der Daten der Datei Fall. An der Stelle gilt Folgendes zu unterscheiden. Wenn das Aufnahmedatum der FALL-Datei mit dem Aufnahmedatum der FAB-Datei übereinstimmt, dann liegt ein Aufnahme-Ereignis vor. Das Selbe betrifft das Entlassungsdatum in den FALL- und FAB-Dateien. Wenn das Aufnahmedatum und das Entlassungsdatum der FAB-Datei nicht mit den Daten der FALL-Datei übereinstimmen, dann liegt eine Verlegung, somit ein neuer Ereignistyp, vor.

Die aufgeführten Datenfelder beziehen sich lediglich auf die drei Ereignistypen. Die restlichen Datenfelder werden bei der Definition weiterer Ereignistypen relevant und werden an der Stelle nicht näher betrachtet. Die Extraktion der Daten aus den nach XML konvertierten CSV-Daten erfolgt durch Einsatz von JAXB. Unabhängig davon, welche Datenfelder ausgewählt werden, wird durch JAXB das Auslesen aller Datenfelder ermöglicht. Weitere Details hinsichtlich der Implementierung werden in Kapitel 7.1 beschrieben. Ein Fall kann in der Regel mehrere OPS oder ICD-Codes haben, wechselt mehrmals die Fachabteilung und aufgrund mehrerer OPS-Codes wird er sicherlich auch mehrere Entgelte haben. In Abbildung 34 wird die Beziehung der §21-Daten in einem Klassendiagramm dargestellt.

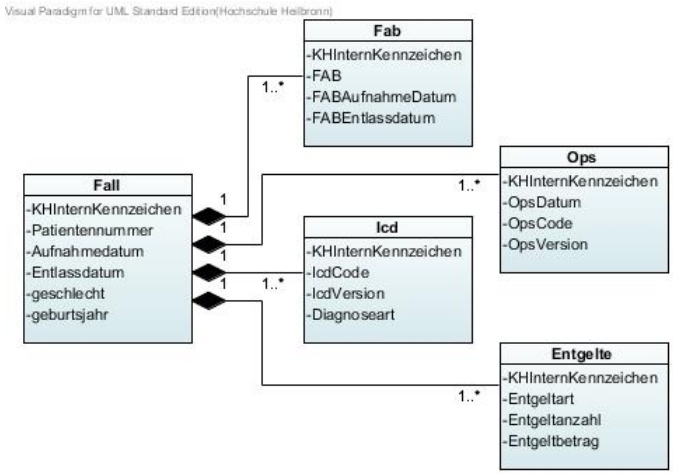


Abbildung 34 §21-Domäne

Zur Simulation der Fallereignisse muss sichergestellt werden, dass die importierten Daten valide sind. Die Vereinbarung der *InEK* definiert im Dokument *Fehlerverfahren* dafür 3 Prüfmodule (Datenlieferung gem. §21 KHEntgG, InEK GmbH - Fehlerverfahren S.5). Für den Einsatz der Daten im Generator sind folgende Prüfmodule relevant.

- Formatprüfung
- Datenprüfung (Plausibilitätsprüfung)

Unter der Annahme, dass die Krankenhäuser sich an die von der *InEK* definierte Vereinbarung halten, müssen die gelieferten Daten hinsichtlich des Formats und der Datentypen valide sein. Der Datenimport verfolgt den Ansatz, das CSV-Format nach XML zu überführen. An der Stelle müssen die XML-Daten zum Einen wohlgeformt sein und zum Anderen gegen ein XML-Schema validiert werden können. Unter diesem Aspekt, dass die Validierung der §21-Daten auf der Vereinbarung der *InEK* aufbaut, wird sie als Referenz zur Validierung der XML-Daten herangezogen. Allerdings muss hier beachtet werden, dass die Validierung der XML-Daten gegen ein XML-Schema lediglich die Datentypen berücksichtigt. Zur Formatprüfung nach der *InEK*-Vereinbarung gehören folgende Aspekte.

- Prüfen der Anzahl der Datenfelder (Spalten) in einer Zeile
- Prüfen auf fehlerhafte Fallnummern
- Prüfen auf Redundanz der Fälle in der Datei FALL

Die Prüfung der drei Aspekte muss explizit durch eigene Implementierungen erfolgen. In Kapitel 7 wird die Typprüfung mit JAXB und Formatprüfung implementierungsspezifisch näher erläutert.

Neben der Typ- und Formatvalidierung müssen die konstruierten Fallereignisse zeitlich konsistent sein. Als Beispiel werden zwei Sachverhalte aufgeführt, die zwar in der Praxis auftreten können, die aber nicht unbedingt als Fehler anzusehen sind. Zum Einen kann es vorkommen, dass bei einem Behandlungsfall das Fallereignis OP nach der Entlassung auftritt. Der Grund hierfür ist die nachträgliche Codierung der OPS-Daten, bei denen die Systemzeit am Tag der Codierung als Default-Wert für eine OP eingetragen und nicht aus dem OP-Bericht entnommen wird. Zum Anderen sprengt die Geburt, als Fallereignis, die Konsistenz der Ereigniskette, indem sie vor der Aufnahme des Neugeborenen auftritt. In dem Fall wird das Neugeborene nachträglich formal aufgenommen. In Kapitel 5.1.1.4 wird eine mögliche Ereigniskette aus Aufnahme, OP und Entlassung dargestellt und die Relevanz des Vorgängers und Abstandes bei einem Fallereignis beschrieben. Die §21-Daten liegen zur Konstruktion der Fallereignisse als Strings vor. Zur Ermittlung der Ereigniskette muss jedes Datenfeld mit einem Datum in ein Datumsobjekt überführt werden. Auch hier wird die benötigte Implementierung in Kapitel 7 aufgeführt.

6. Design

Die Anforderungsanalyse ermittelt die eigentlichen Objekte des Anwendungsbereiches. Sie werden in der Softwareentwicklung häufig als Domainobjekte bezeichnet. Domainobjekte stellen Daten zur Verfügung und können über Funktionen manipuliert werden. Dieser Sachverhalt wird durch Klassen abstrahiert.

6.1. Domänenmodell

Das Domänenmodell definiert hinsichtlich der Java-EE-Schichtenarchitektur die Datenschicht und bildet somit bis auf die Datenbank die unterste Schicht ab. In Kapitel 5.4 wurde bereits die §21-Domäne als Klassendiagramm dargestellt sie bezieht sich ausschließlich auf §21-Daten. In Abbildung 35 wird das Domainmodell des Generators vorgestellt, welches durch die Klasse Behandlungsfall eine Verbindung zur §21-Domäne herstellt.

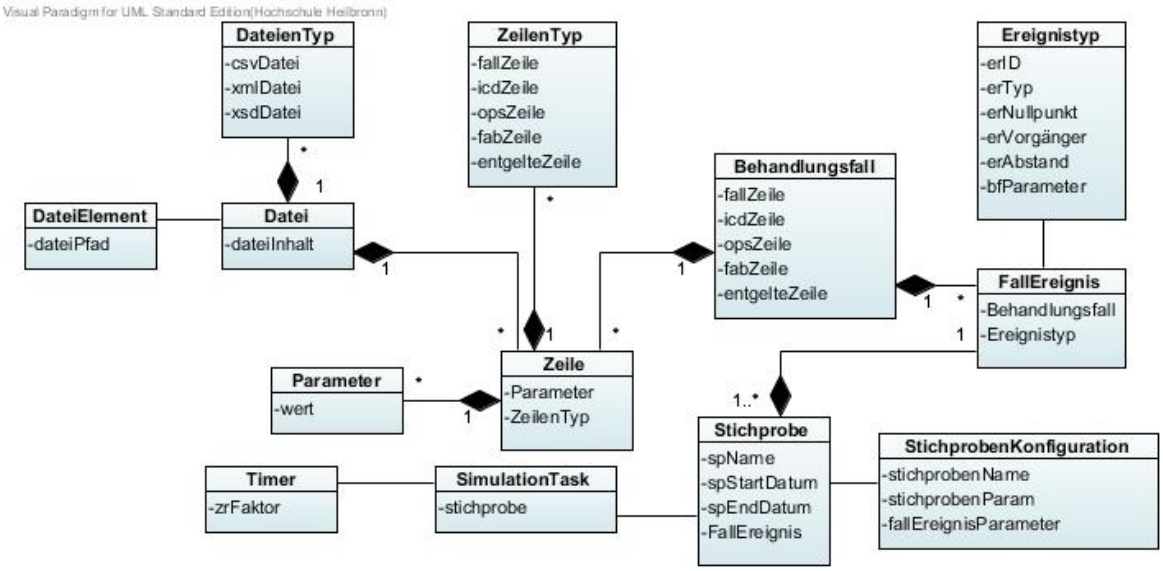


Abbildung 35 Domainmodell

Die §21-Daten werden in Dateien gehalten, die als Dateielemente in einem Verzeichnis vorliegen. Im Zusammenhang mit der Konvertierung nach XML definiert eine Datei einen Dateityp. Dieser kann eine CSV-, XML- oder XML-Schema-Datei sein. Jede CSV-Datei definiert eine Datenstruktur, die durch Zeilen repräsentiert wird. Abhängig von der CSV-Datei wird jede Zeile durch einen Zeilentyp repräsentiert und hat entsprechende Parameter mit Werten. Die fünf CSV-Dateien aus Kapitel 5.4 definieren die §21-Domäne und werden durch JAXB als Klassen generiert. Die generierten Klassen werden im Klassendiagramm in Abbildung 35 durch die Beziehung zwischen ZeilenTyp und Zeile dargestellt. Abhängig vom Zeilentyp wird jede Zeile durch die Klasse Behandlungsfall aggregiert. Jeder Behandlungsfall hat im Verlauf einer Behandlung Ereignisse. Jedes Fallereignis wird durch einen Ereignistypen spezifiziert. Die Fallereignisse werden anhand einer Stichprobe weiter spezifiziert. Jede Stichprobe wird als Stichprobenkonfiguration dateibasiert abgelegt. Die Stichproben mit Fallereignissen werden zur Simulation dem Simulationstask übergeben. Der Simulationstask wird durch einen Timer getriggert.

6.2. Datenzugriff

Die Datenzugriffsschicht definiert in der Regel CRUD-Funktionen. In Kapitel 4.1.1 wurden mögliche Alternativen hinsichtlich des Datenzugriffs aufgeführt. Der konzeptuelle Ansatz definiert den Einsatz von JAXB, JDO und als Backend die objektorientierte Datenbank db4o.

Für den Zugriff auf die §21-Daten werden die CSV-Daten nach XML konvertiert. Diese Funktionalität wird in Kapitel 7.1.1 näher erläutert. Mit JAXB, siehe Kapitel 3.4.1, wird die §21-Domäne generiert und somit das Auslesen der §21-Daten aus den XML-Dateien ermöglicht. Das Auslesen der §21-Daten geschieht in JAXB durch die *unmarshall*-Methode. In der Abbildung 36 wird dazu das *IJaxbDao*-Interface definiert, das Methoden zum Auslesen des jeweiligen Zeilentyps aus der XML-Datei und Speichern in eine db4o-Datenbank zur Verfügung stellt. Für das Ablegen der Objekte in db4o wird zunächst der Ansatz von JDO verfolgt, indem die zughörigen Klassen für das Enhancement, siehe Kapitel 3.4.2 präpariert werden. Einschränkend muss gesagt werden, dass der Einsatz von JDO nur bis zu der Stelle funktioniert hat, an der mit der *persist*-Methode auf die db4o-Datenbank zugegriffen wird. Ein Ablegen

von Objekten in der Datenbank durch JDO war nicht möglich. Trotz ausgiebiger Fehlersuche konnte die Ursache dafür nicht ermittelt werden. Aus diesem Grund wird db4o direkt über die selbst definierte *Db4oDao*-Klasse, siehe Kapitel 7.1.2, angesprochen. Des Weiteren muss hinzugefügt werden, dass alternativ der Ansatz, die Objekte als Behandlungsfall im Arbeitsspeicher zu kapseln, bei der Datenmenge zu Performanceeinbußen führen würde. Zum Einen, weil das Binding von JAXB modellbasiert arbeitet, siehe Kapitel 3.3.3 und zum Anderen, weil jeweils für eine XML-Datei nur eine Instanz des *Unmarshallers*, siehe Kapitel 3.4.1, möglich ist.

Für den Zugriff auf die Behandlungsfälle wird das *BehandlungsfallDao*-Interface definiert, welches die Methoden mit CRUD-Charakter anbietet, wobei die *create*- und *update*-Methode zur *createUpdate*-Methode zusammengefasst werden. Die createUpdate-Methode dient zum Aggregieren der Objekte aus §21-Domäne und zum Wieder-Ablegen dieser als Behandlungsfälle in der db4o-Datenbank. Die Fallereignisse definieren die Verknüpfung des Ereignistyps und des Behandlungsfalls und werden zur Simulationszeit als XML-Dateien geschrieben. Demnach werden sie nicht in der Datenbank abgelegt, sondern in der Serviceschicht verarbeitet. Die Verwaltung der Ereignistypen erfolgt durch Methoden aus dem *EreignisTypDao*-Interface. In der Abbildung 36 wird die DAO-Schicht prototypspezifisch dargestellt. Im Hinblick auf das Konzept käme zusätzlich das *StichprobenDao*-Interface hinzu, welches die Methoden zur Verwaltung der Stichproben anbietet. Wobei die Stichprobe wiederum dateibasiert gespeichert wird.

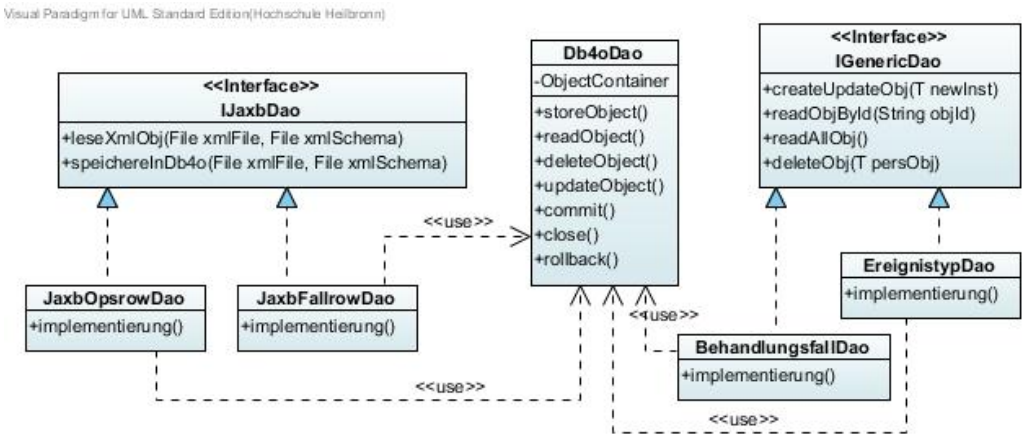


Abbildung 36 DAO-Schicht des Prototyps

6.3. Serviceschicht

Die Serviceschicht definiert Funktionen, die zur Laufzeit auf Daten operieren, die über die Datenzugriffsschicht zur Verfügung gestellt werden. Für den Prototyp werden folgende in der Abbildung 37 dargestellte Klassen definiert. Zwecks der Entkopplung werden für die Klassen Interfaces definiert. Zum *SimulationService*-Interface gehören Methoden, die den Ereignistyp und den Behandlungsfall in Beziehung setzen und als Fallereignis zurückgeben. Desweiteren wird durch das Starten der Simulation die Abarbeitung der zeitlichen Reihenfolge der Fallereignisse gestartet. Je nach dem welchen Ereignistyp das Fallereignis mitführt, wird es dem entsprechenden Timertask zugeordnet. Der Timertask ruft in Zeitabständen die *schreibeXMLEreignis*-Methode auf um die Fallereignisse zu schreiben. Für Berechnungen der Zeitabstände müssen zunächst die Datumangaben in Datumsobjekte geparst werden, weil sie in der Datenbank als Strings vorliegen. Dafür wird im *IUtil*-Interface die entsprechende Methode angeboten. Zur Berechnung des Zeitabstandes wird die *berechneZeitspanne*-Methode angeboten, die den Abstand unter den Fallereignissen ermittelt, welcher für den Timertask als Wert in Millisekunden übergeben wird. Die *ermittleDauer*-Methode ermittelt die Zeit, die das System braucht um einen Vorgang auszuführen. Sie wird beispielsweise zur Messung der Schreibdauer für XML-Dateien verwendet um den Wert der Zeitkomponente zur Verfügung stellen zu können.

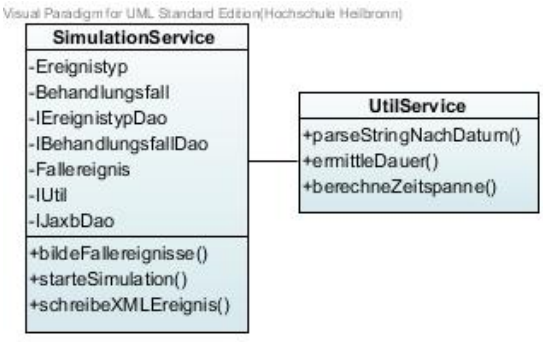


Abbildung 37 Service-Schicht des Prototypen

Hinsichtlich des Generator-Konzeptes müssen weitere Interfaces deklariert werden, die den gesamten Geschäftsprozess des Generators nachbilden. An der Stelle muss die Granularität des Geschäftsprozesses berücksichtigt werden, siehe Kapitel 3.2. Es entsteht ein Geflecht aus Service-Interfaces, die im Zusammenhang mit Jax-WS,

siehe Kapitel 3.4.3, durch die Annotation *@Webservice* zur Webservice-Schicht definiert werden, siehe Kapitel 7.1.4.

6.4. Präsentationsschicht

Die Präsentationsschicht stellt Funktionen zur Verfügung, die die Eingaben des Benutzers verarbeiten. Konzeptuell wird für die Oberfläche ein GUI-Mockup entwickelt, der im Pflichtenheft im Detail beschrieben wird.

Den Einstieg definiert das *Dispatcher-Servlet* von Spring. Es regelt die HTTP-Anfragen und wird in Kapitel 3.4.4 im Zusammenhang mit Spring-MVC näher erläutert. Für eine detaillierte Beschreibung eines http-Requests wird auf (Wolff 2009 S.274) verwiesen. Des Weiteren wird eine Anwendungskontext-Datei, auch *Application-Context* genannt, benötigt. In dieser Datei erfolgt die Konfiguration der Spring-Beans, die unter Anderem Controller und Views registrieren. In der Abbildung 38 werden ansatzweise zwei Views definiert, die eine regelt die Aufnahme des Ereignistyps und die andere zeigt die aufgenommenen Ereignistypen an. Alle Eingaben des Benutzers in einem Formular, wie beispielsweise *EreignistypView*-Formular, müssen überprüft werden. Hierfür wird eine *Validator*-Klasse definiert. Bei der Verarbeitung der Eingaben greift der *Ereignistyp*-Controller auf den Validator zu, bevor die Daten als Command-Objekt der Serviceschicht übergeben werden. Das Command-Objekt ist in dem Fall das Pendant zu den DTOs, siehe Kapitel 4.1.1.4.

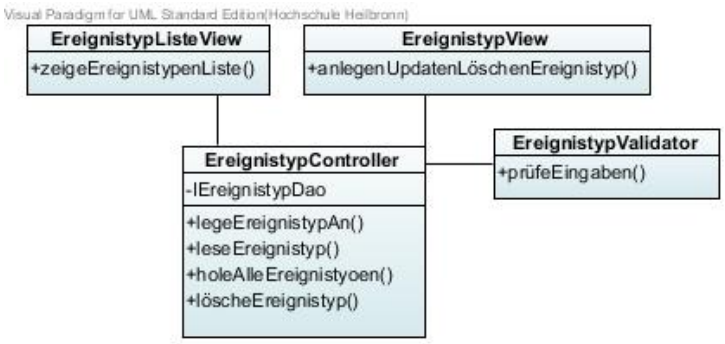


Abbildung 38 Präsentationsschicht des Prototypen

Das Entwickeln weiterer Controller und Views geschieht analog wie zuvor beschrieben. Den Ansatz liefert das GUI-Mockup aus dem Pflichtenheft. Abhängig von der Implementierung können für jede Komponente des Generators ein oder mehrere Controller und zugehörige Views definiert werden. Durch die Spring-Beans – Handler-Mapping und View-Resolver – werden die Controller und Views angesprochen.

6.5. Zusammenhang über Dependency Injection

Die einzelnen Schichten werden im Zusammenhang mit Spring über den *Application-Context* in Beziehung gesetzt, siehe Kapitel 3.4.4. Das Framework Spring kümmert sich um die Instanzierung und Konfiguration der Objekte. In Abbildung 39 wird der Zusammenhang vereinfacht dargestellt.

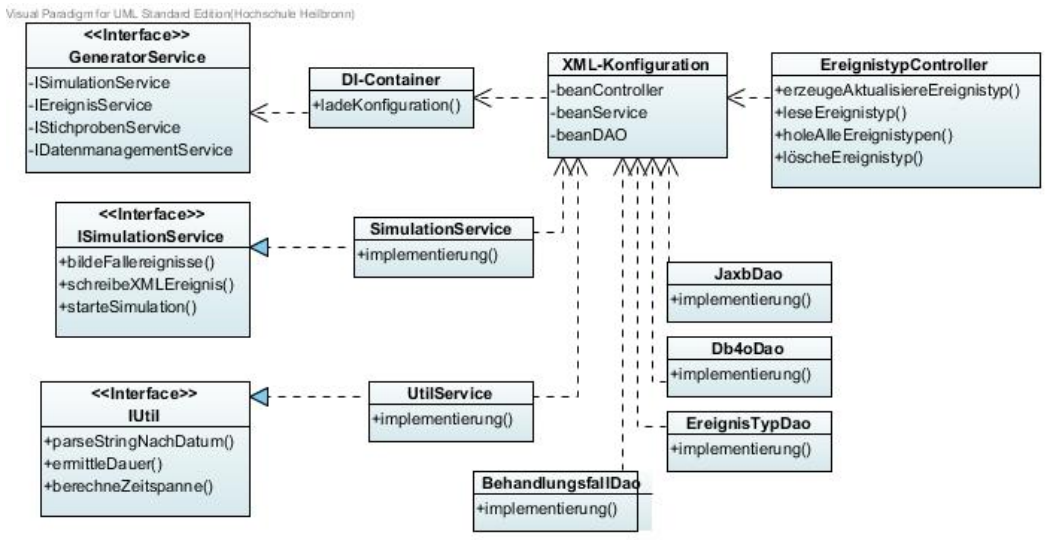


Abbildung 39 Dependcy Injection im Generator

Die Verschaltung der Beans erfolgt in der XML-Konfiguration, die den Anwendungskontext spezifiziert. Die Beans werden mit einer ID versehen und zur Laufzeit über die Setter-Methoden injiziert. Weitere Informationen zur Handhabung der Konfiguration sind in (Wolff 2009; Walls & Breidenbach 2007) zu finden.

7. Implementierung und Test

In diesem Kapitel werden die Realisierung der an den Prototyp spezifizierten Anforderungen sowie der Implementierungsansatz hinsichtlich der Tests beschrieben.

7.1. Implementierungsansatz

Im STS werden implementierungsspezifisch folgende Projekte aufgesetzt. Das *Jaxb-Datenmanagement*-Projekt definiert ein Ant[11]-Script zur Generierung der §21-Domäne. Das *JDO-Datenmanagement*-Projekt wird im Zusammenhang mit db4o aufgesetzt die Persistenz-Technologie nachzubilden. Im Bezug auf die Problematik mit JDO wird alternativ im *DB4O-Datenmanagement*-Projekt die Verwaltung der Objekte in der db4o-Datenbank ohne JDO definiert. Im Projekt *Generator-Datenmanagement* ist der Ansatz zur Konvertierung der CSV-Daten nach XML zu finden. Die Implementierungsansätze sind in entsprechenden Projekten im *workspace_da,* im Ordner *Entwicklung* auf dem beigelegten Datenträger zu finden.

7.1.1. Konvertierung

Die Konvertierung wird durch die *CSVToXMlConverter*-Klasse definiert. Sie liest zeilenweise die CSV-Daten aus und schreibt sie wieder als XML-Elemente dateibasiert auf den Datenträger. Dazu wird aus Kapitel 2.3 die Analyse der CSV-Dateien herangezogen. Der entscheidende Schritt ist die Zerlegung der Datenfelder, die durch das Semikolon getrennt werden. In Java wird dafür die *split*-Methode angeboten, jedoch ist an der Stelle zu beachten, dass die Nutzung der *split*-Methode mit Einschränkungen möglich ist. Zum Einen dürfen die CSV-Daten keine leeren Felder enthalten und zum Anderen darf das Trennzeichen nicht innerhalb eines Datenfeldes auftauchen. Trotz der Vereinbarung der *InEK* werden, in den Testdaten

[11] Ant: Java-basierte XML-Sprache zur Erstellung von Build-Skripten. Zum Build gehört Kompilieren, Packen und Verteilen von Quellcode.

von 2006, leere Datenfelder mitgeführt. Zur möglichst genauen Trennung der Daten wird die, in (Louis & P. Müller 2007 S.262) definierte *CSVTokenizer*-Klasse verwendet, die Methoden für das Lesen und Parsen der CSV-Daten zur Verfügung stellt. Diese Klasse wird in der *erzeugeXML-Mehode* der *CSVToXMLConverter*-Klasse eingesetzt. In der Abbildung 40 wird der Ausschnitt des Konvertierungsvorgangs dargestellt.

```
private void erzeugeXml(File fileDummy, String zielPfad) {

            elRoot = new Element(fileName.toLowerCase());

            // CSV-Datei oeffnen...
            csv = new CSVTokenizer(fileDummy.getAbsolutePath(), ';');

            // Ueberschriften einlesen und als Knoten verwenden...
            header = csv.nextLine();

            // Zeilen einlesen und als XML-Knoten ausgeben...
            while (csv.hasMoreLines()) {

                  // KnotenElement dem RootElement hinzu
                  elKnoten = new Element(rowDummy.concat("row"));
                  elRoot.addContent(elKnoten);

                  fields = csv.nextLine();

                  // Felder als XML-Knoten ausgeben...
                  for (int i = 0; i < header.length; i++) {

                        attKnoten = new Attribute(header[i].trim(), fields[i]);
                        elKnoten.setAttribute(attKnoten);
                  }
            }
            xmlDoc = new Document().setContent(elRoot);
            Format format = Format.getPrettyFormat();
            format.setEncoding("ISO-8859-1");
            XMLOutputter xmlOut = new XMLOutputter(format);
            xmlOut.output(xmlDoc, out);
}
```

Abbildung 40 Code-Ausschnitt zur Konvertierung von CSV nach XML

Ein weiterer wichtiger Aspekt ist hinsichtlich der Kodierung der CSV-Daten, die nach XML konvertiert werden, zu beachten. Die Bezeichnungen der Datenfelder in den CSV-Daten enthalten Umlaute. Beim Schreiben der XML-Dateien in der UTF-8-Kodierung wird eine *jdomIllegalNameException* geworfen, weil das Umlaut-Zeichen im UTF-8 nicht korrekt abgebildet wird. Stattdessen muss die ISO-8859-1-Kodierung gewählt werden. Desweiteren muss die Entwicklungsumgebung hinsichtlich der Textkodierung ebenfalls auf ISO-8859-1 gesetzt werden. Als Ergebnis liegen die §21-Daten im XML-Format vor.

Den nächsten Schritt definiert die Generierung der §21-Domäne mit JAXB. Dafür wird zunächst jeweils für eine XML-Datei ein XML-Schema abgeleitet. Die Ableitung kann

manuell erfolgen, indem Elemente und Attribute der XML-Datei explizit durch Entwickler, hinsichtlich der Struktur und Datentypen, definiert werden. Dieser Schritt ist sehr fehleranfällig, weil die Struktur der XML-Dateien jeweils für ein Datenfeld exakt nachgebildet werden muss. Aus diesem Grund wird das Entwicklertool Trang eingesetzt, das jeweils aus einer XML-Datei ein XML-Schema ableitet. In der Abbildung 41 wird der Vorgang veranschaulicht.

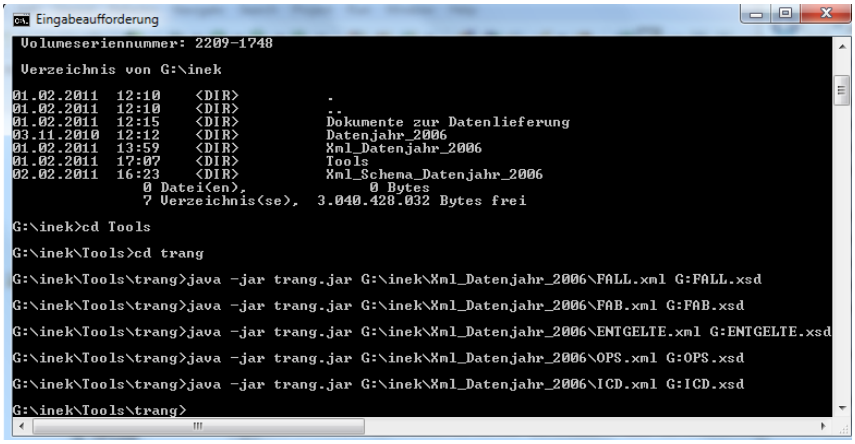


Abbildung 41 Ableiten des XML-Schemas mit Trang

In der Eingabeaufforderung wird der *–jar*-Befehl in dem Verzeichnis ausgeführt, in dem die trang.jar liegt. Als Paramater werden das Quell- und Zielverzeichnis für die XML-Datei und das XML-Schema übergeben. In der Abbildung 41 wird der Vorgang dargestellt. Alternativ ist es möglich ein XML-Schema für alle fünf XML-Dateien[12] abzuleiten. Allerdings wird zum Einen schnell die Struktur des XML-Schemas unübersichtlich und zum Anderen wird für das Einlesen und Validieren lediglich einer XML-Datei das gesamte XML-Schema eingesetzt. Für die Generierung der §21-Domäne mit JAXB spielt es keine Rolle, ob das XML-Schema vereinzelt oder zusammengefasst, abgeleitet wird.

Der Einsatz von Trang nimmt hinsichtlich der Definition von XML-Schemadokumenten zwar viel Arbeit ab. An der Stelle ist jedoch hinzuweisen, dass einige Datentypen nicht exakt erkannt werden. In dem Fall muss das XML-Schema an einigen Stellen angepasst werden. In der Entwicklungsumgebung ist es möglich Änderungen am XML-Schema vorzunehmen. Dafür bietet sich der Design-Modus an. In der Abbildung 42 wird beispielsweise die OPS.xsd grafisch dargestellt.

[12] Fünf XML-Dateien definieren die medizinischen Falldaten, bestehend aus FALL, FAB, ICD, OPS, ENTGELTE.

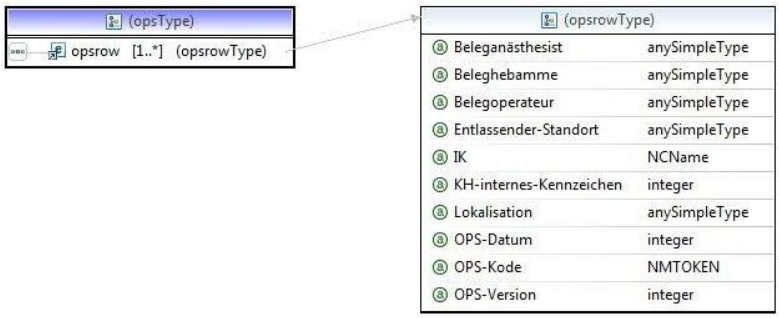


Abbildung 42 OPS.xsd im Designmodus

Die Datenfelder IK und OPS-Code werden, in der Abbildung 42, durch den Trang-Einsatz nicht korrekt erkannt. In dem Fall muss hinsichtlich des Datentyps vereinzelt nachgebessert werden. In der Vereinbarung der *InEK,* siehe Externe Dokumente, werden die Datentypen der Datenfelder aufgelistet und können als Referenz herangezogen werden.

Zur Generierung der §21-Domäne wird ein Schema-Compiler benötigt, der aus den XML-Schemata die Java-Klassen erzeugt. Der Schema-Compiler wird in einem Ant-Script als xjc-Task definiert, der von der JAXB-API zur Verfügung gestellt wird. Des Weiteren werden die Quell- und Zielverzeichnisse für die XML-Schemata und die generierten Java-Klassen definiert. Das Ant-Script beschreibt den eigentlichen Generierungsprozess und wird in der Abbildung 43 als Code-Ausschnitt dargestellt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project default="rebuild" name="Generator-Datenmanagement">

	<path id="path.lib.jaxb">
		<fileset dir="..//lib" includes="**/*.jar" />
	</path>

	<property name="dir.src" location="../src-jaxb/xmlschema" />
	<property name="dir.srcgen" location="../src-jaxb/" />
	<property name="dir.bin" location="../bin" />

	<target name="-xjc">

		<!-- Task definieren -->
		<taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask"
			classpathref="path.lib.jaxb" />

		<!-- Binding Compiler Schema->Java starten -->
		<xjc destdir="${dir.srcgen}">
			<schema dir="${dir.src}" includes="**/*.xsd" />
			<produces dir="${dir.srcgen}" />
		</xjc>
	</target>
</project>
```

Abbildung 43 Ant-Script zur Generierung der §21-Domäne

7.1.2.Persistenz

In Kapitel 3.4.2 wird hinsichtlich der Persistenz der Daten das JDO-Framework in Betracht gezogen. Zur Konstruktion der Fallereignisse werden Daten aus der §21-Domäne verwendet, die zunächst als Behandlungsfall gekapselt werden. Desweiteren werden Ereignistypen benötigt, die ein Fallereignis spezifizieren. Mit JDO wird zunächst der Ansatz verfolgt, die Behandlungsfälle und Ereignistypen in eine objektorientierte Datenbank, indem Fall db4o, abzulegen. Für den Zugriff auf db4o werden Bibliotheken des Persistenz-Frameworks – Datanucleus[13], benötigt, der die Transparente Persistenz durch das Enhancement der POJO-Klassen regelt. Für die Entwicklungsumgebung wird in dem Fall das Datanucleus-Plugin angeboten, das den Enhancement-Prozess automatisiert. Der Entwickler kann dadurch aus Klassen, persistiert werden, die Meta-Informationen ableiten und die kompilierten Java-Klassen mit Bytecode erweitern lassen, siehe Kapitel 3.4.2. Das Persistenz-Problem im Zusammenhang mit JDO wurde bereits 6.2 angeführt. Der Grund dafür könnte sein, dass die *persist*-Methode optional angeboten wird und im Zusammenspiel mit db4o nicht unterstützt wird. Trotz der Recherchen kann dieses Problem in der dafür eingeplanten Zeit nicht behoben werden und wird an der Stelle zurückgestellt.

Das Framework JDO im Zusammenhang mit Datanucleus definiert zwar den Datenzugriff auf db4o. Es ist aber zur Prototypentwicklung nicht zwingend notwendig. Der Zugriff auf die db4o-Datenbank erfolgt über die Instanzierung der *ObjectContainer*-API. Die *Db4oDao*-Klasse in Kapitel 6.2 definiert Methoden, die zur Verwaltung der Objekte in db4o regeln. Die einfachste Art auf db4o zuzugreifen, ist eine Datendatei als Objekt-Container über die *openFile(config, file)*-Methode bereitzustellen. Der Parameter *file* definiert den Pfad (absolut oder relativ) der Datei für die db4o-Datenbank. Beim ersten Zugriff wird die Datei erzeugt und bei weiteren Zugriffen mit der Datei gearbeitet. Der Parameter *config* beschreibt die Konfiguration der db4o-Datenbank, die über *Db4oEmbedded.newConfiguration()* instantiert werden kann. Die Konfiguration der Datenbank muss vor dem Aufruf der *openFile()*-Methode erfolgen. Durch den Aufruf der *openFile()*-Methode wird der *ObjectContainer* zurückgegeben, der den DAO-Klassen übergeben werden kann, zur weiteren Bearbeitung von Objekten. Jedes Objekt wird durch die *store(Object)*-Methode persistiert und durch die *commit()* wird die Transaktion beendet. Analog zu den

[13] DataNucleus: Persistenzframework hinsichtlich relationaler-, xml- oder objektorientierter Speicherung von Daten. Es setzt zur Abbildung der Daten auf das Backend JDO- oder JPA-Mapper ein, als Backend kann unter Anderem db4o eingesetzt werden.

relationalen Datenbanken werden die Transaktionen in db4o nach dem ACID-Prinzip[14] (Atomicity, Consistency, Isolation Durability) gehandelt. In der Abbildung 44 wird ein Ausschnitt der *Db4oDao*-Klasse vorgestellt.

```
private static String DB4OFILENAME = System.getProperty("user.home")
            + "/db4oTest.db";

public static void store(Object object) {}
public static void delete(Object object) {}     }

public static <T> List<T> query(Predicate<T> predicate) {
      return getObjectContainer().query(predicate);
}

public static <T> List<T> query(Predicate<T> predicate,
            Comparator<T> comparator) {
      return getObjectContainer().query(predicate, comparator);
}

public static void refresh(Object object) {
      getObjectContainer().ext().refresh(object, 1);
}

public static ObjectContainer getObjectContainer() {
      if (objectContainer == null) {
            EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();
            config.common().activationDepth(4);
            config.common().updateDepth(2);
            config.common().automaticShutDown(false);
            objectContainer = Db4oEmbedded.openFile(config, DB4OFILENAME);
      }
      return objectContainer;
}
```

Abbildung 44 Ausschnitt aus der Db4oDao-Klasse

Die *Db4oDao*-Klasse wird unter Anderem zur Speicherung der Behandlungsfälle und Ereignistypen in der db4o-Datenbank verwendet. Aufgrund der §21-Datenmenge muss eine Datenstruktur definiert werden, die zur Laufzeit die §21-Daten der §21-Domäne als Behandlungsfall kapselt. Bei der Generierung der §-21-Domäneklassen wird die Beziehung der Datei zu Zeile für jeweiligen Zeilentyp, siehe in Kapitel 6.1 die Abbildung 35, nach zwei Klassen aufgelöst. Jede Zeile mit den Datenfeldern, beispielsweise der FALL-XML-Datei, wird als *Fallrow*-Klasse und der Rumpf als *Fall*-Klasse deklariert. Im *IJaxbDao*-Interface, siehe Kapitel 6.2 werden entsprechende Methoden definiert, die den jeweiligen Zeilentyp der XML-Datei lesen und in der db4o-Datenbank speichern. In der Abbildung 45 werden die beiden Methoden dargestellt, die beispielsweise die FALL.xml auslesen und in die db4o-Datenbank ablegen.

[14] ACID: Es beschreibt erwünschte Eigenschaften von Transaktionen in Datenbankmanagementsystemen.

```
public List<Fallrow> readFallXml(File fallXml, File fallSchema) {

            try {
                  con = JAXBContext.newInstance(Fall.class);
                  unm = con.createUnmarshaller();
                  fall = (Fall) unm.unmarshal(fallXml);
            } catch (JAXBException e) {
                  e.printStackTrace();
            }

            try {
                  schema = sf.newSchema(fallSchema);
            } catch (SAXException e) {
                  e.printStackTrace();
            }
            unm.setSchema(schema);

            List<Fallrow> fallList = new ArrayList<Fallrow>();
            for (Fallrow fallrow : fall.getFallrow()) {
                  fallList.add(fallrow);
            }
            return fallList;
}

public void persistFallToDb4o(Fallrow fallrow, File fallXml,
                        File fallSchema) {
            Iterator<Fallrow> iterator = readFallXml(fallXml, fallSchema)
                        .iterator();
            try {
                  new File("db4oTest.db").delete();

                  while (iterator.hasNext()) {
                        fallrow = iterator.next();
                        Db4oDao.store(fallrow);
                        Db4oDao.commit();
                  }
            } finally {
                  Db4oDao.close();
            }
}
```

Abbildung 45 Methoden zum Lesen der FALL.xml und Speichern in db4o

Zum Einlesen der §21-Daten wird in der *readFallXml*-Methode eine *Unmarshaller*-Instanz, siehe in Kapitel 3.4.1 erzeugt, die von dem *JAXBContext* zur Verfügung gestellt wird. Mit der *getFallrow*-Methode aus der Fall-Klasse werden die Zeilen der FALL-XML-Datei eingelesen und als Liste vom Typ *Fallrow* zurückgegeben. Das Ergebnis wird der *persistFallToDb4o*-Methode zur Verfügung gestellt und durch den Aufruf der Verwaltungsmethoden der *Db4oDao*-Klasse in der db4o-Datenbank gespeichert. Analog können die restlichen §21-Daten der FAB-, ICD-, OPS- und ENTGELTE-XML-Dateien eingelesen und persistiert werden. Es ist durchaus möglich bereits beim Einlesen der §21-Daten die Behandlungsfälle zu kreieren. Allerdings werden dafür mehrere *Unmarshaller*-Instanzen benötigt, die ohne Weiteres nicht zur Verfügung stehen. Laut der JAXB-API darf nur eine *Unmarshaller*-Instanz erzeugt werden. Für das Vorhaben mehrere *Unmarshaller*-Instanzen zu erzeugen, wird der *UnmarshallHandler* zur Verfügung gestellt. Der *UnmarshallHandler* baut jedoch auf den SAX-Parser auf und generiert zusätzlichen Aufwand, weil der Entwickler

zusätzliche Handler- oder Callback-Klassen zur Verfügung stellen muss, siehe in Kapitel 3.3.3. Folglich werden die §21-Daten vereinzelt in die db4o-Datenbank eingelesen[15].

Zur Konstruktion der Fallereignisse müssen die Behandlungsfall- und Ereignistyp-Objekte in der db4o-Datenbank angefragt werden. Dafür werden als Anfragesprachen QBE (Query By Example)-, Native- und SODA (Simple Object Data Access)-Query zur Verfügung gestellt. Die QBE-Query definiert für eine Anfrage ein Beispielobjekt, welches anhand seiner Parameter in der Datenbank angefragt wird. Die QBE-Query ist die Einfachste von den drei vorgestellten Anfragesprachen. Sie ist jedoch eingeschränkt nutzbar, weil unscharfe oder bedingte Anfragen in Objektstrukturen nicht möglich sind. Die Native-Query wird in der Regel zur Laufzeit analysiert und im Erfolgsfall in eine SODA-Query übersetzt. Die SODA-Query erstellt einen Objektgraph, in dem Objekte anhand eines Kriteriums für das Anfrageergebnis eingeschränkt werden. Die SODA-Query ist die mächtigste Anfragesprache, allerdings schwer zu erstellen. Sowohl bei den Native- als auch bei den SODA-Queries müssen die Anfragen hinsichtlich der Komplexität wohlüberlegt sein. Weitere Details zu den Anfragesprachen im Zusammenhang mit db4o können in der Literatur (Paterson u. a. 2006), sowie auf der Homepage von db4o (db4o Reference Dokumentation) nachgelesen werden.

7.1.3.Zeitkomponente

Die Zeitkomponente definiert Funktionalitäten zur Simulation der Fallereignisse. Dazu gehören zum Einen die Konsistenz der zeitlichen Abfolge – die Ereigniskette und zum Anderen der Bezug zum jeweiligen Behandlungsfall. In Kapitel 5.1.1.4 wird dieser Sachverhalt aufgeführt. Unter der Annahme, dass ein Fallereignis gebildet werden kann, müssen zunächst folgende Berechnungen durchgeführt werden. Aus den Behandlungsfällen können die Daten bezüglich des Datums extrahiert werden. Der Ereignistyp liefert Informationen zum Ereignistyp, Nullpunkt, Vorgänger und Abstand. Die Ereigniskette in Kapitel 4.1.1.4 definiert Vorgaben, nach denen die Ereignistypen auftreten sollen. Der Nullpunkt wird bei beim Anlegen des Ereignistyps definiert und

[15] Zum aktuellen Zeitpunkt können die Behandlungsfälle nicht ohne Weiteres zusammengeführt werden.

hat keinen Vorgänger. Demnach wird der Vorgänger bei dem Ereignistyp Aufnahme auf den Nullpunkt gesetzt. Hinsichtlich der OP- und Entlassungs-Ereignisse müssen die Vorgänger jeweils angepasst und der zugehörige Abstand, bzw. die zeitliche Differenz, neu berechnet werden. Der Ereignistyp OP benötigt neben dem OPS-Datum zusätzlich das Datenfeld OPS-Code. Der OPS-Code wird auf die erste Stelle des Zeichensatzes untersucht und wenn es der Zahl 5 entspricht werden die Parameter für das OP-Fallereignis gesetzt. In der Abbildung 46 wird ansatzweise die *bildeFallereignisse*-Methode dargestellt, die für ein Fallereignis den Vorgänger und den Abstand ermittelt und als Rückgabewert eine Liste mit Fallereignissen zurückgibt, die jeweils einen Abstand und Vorgänger mitführen.

```
public List<Fallereignis> bildeFallereignisse() {
            ...
            while (ertIt.hasNext()) {
                  ert = ertIt.next();
                  while (bfIt.hasNext()) {
                        bf = bfIt.next();
                        Iterator<Opsrow> opsRows = bf.getOpsrows().iterator();
                        if (fe == null) {
                              fe = new Fallereignis();
                              if (ert.getEreignisVor() == null) {// wenn kein Vorgänger
                                                                 // dann Aufnahme
                                    long abstand = util.berechneZeitspanne(
                                                ert.getEreignisNullpunkt(), bf.getFallrow()
                                                      .getAufnahmedatum().toString());
                                    ert = new Ereignistyp();
                                    ert.setEreignisVor(null); //
                                    ert.setEreignisAbstand(abstand);
                                    fe.setBf(bf);
                                    fe.setErTyp(ert);
                                    feList.add(fe);

                              } else if (ert.getEreignisID() == "02") {
                                    while (opsRows.hasNext()) {
                                          String opCode = opsRows.next().getOPSKode();
                                          long abstand = util.berechneZeitspanne(
                                                      bf.getFallrow().getAufnahmedatum()
                                                            .toString(), opsRows.next()
                                                            .getOPSDatum().toString());
                                          int pos = opCode.indexOf('5');
                                          if (pos == 1) { // wenn opsCode an erster Stelle 5
                                                            // dann setze Parameter
                                                ert.setEreignisVor("01"); //Vorgänger = Aufnahme
                                                ert.setEreignisAbstand(abstand);
                                                fe.setBf(bf);
                                                fe.setErTyp(ert);
                                                feList.add(fe);
                                          } else
                                                return null;
                                    }
                              } else if (ert.getEreignisID() == "03") {
                                    //berechne abstand zwischen OP und Entlassung
                                    //setze Abstand und Vorgänger auf OP

                              } else {
                                    //berechne Absatnd zwischen Aufnahme und Entlassung
                                    //setze Abstand und Vorgänger auf Aufnahme
                  ...
            }
            return feList;
      }
```

Abbildung 46 Codeausschnitt zur Bildung der Fallereignisse

Zur Berechnung der zeitlichen Differenzen müssen die Datumsangaben aus den §21-Daten vom Datentyp *java.util.Date* sein. Dazu werden zwei Methoden definiert. Die *parseStringToDate*-Methode parst die Datumangaben aus den §21-Daten vom Typ *String* nach *Date* und gibt als Rückgabewert das Datums-Objekt zurück, siehe Abbildung 47.

```
public Date parseStringToDate(String stringDate) {
            Date date = null;
            SimpleDateFormat sdfToDate = new SimpleDateFormat("yyyyMMddHHmmss");
            try {
                  date = sdfToDate.parse(stringDate);
            } catch (ParseException e1) {
                  e1.printStackTrace();
            }
            return date;
      }
```

Abbildung 47 Methode parst das Datum vom Typ String in ein Datum vom Typ Date

Die *parseStringToDate*-Methode wird bei der Berechnung der zeitlichen Differenzen in der *berechneZeitspanne*-Methode eingesetzt, siehe Abbildung 48.

```
public long berechneZeitspanne(String datum1, String datum2){
      long abstandMillis = 0;
      Date dat1 = parseStringToDate(datum1);
      Date dat2 = parseStringToDate(datum2);
      abstandMillis = dat2.getTime() - dat1.getTime();
      return abstandMillis;
}
```

Abbildung 48 Methode zur Berechnung der Zeitdifferenz

Desweiteren müssen das Schaltjahr, sowie die Zeitumstellung berücksichtigt werden, denn diese Aspekte haben einen Einfluss auf den Abstand der Fallereignisse. Zur Berücksichtigung der Zeitumstellung und der Schaltjahre muss die *berechneZeitspanne*-Methode in Abbildung 48 erweitert werden. Einen Ansatz zur Berücksichtigung der Zeitumstellung sowie des Schaltjahres liefert die Implementierung der *TimeSpan*-Klasse in Literatur (Louis & P. Müller 2007 S.137). Hinsichtlich der Berechnung der Feiertage wird ebenfalls in (Louis & P. Müller 2007) ein Ansatz aufgeführt.

Für den Schreibvorgang der Fallereignisse ist die *schreibeXMLEreignis*-Methode zuständig. Intern wird eine Marschaller-Instanz von JAXB erzeugt, die das übergebene Fallereignis als XML-Datei schreibt, siehe Abbildung 49.

```
public Fallereignis schreibeXMLEreignis(Fallereignis fe) {
            JAXBContext jaxbCon;
            try {
                  jaxbCon = JAXBContext.newInstance(Fallereignis.class);
                  Marshaller m;
                  m = jaxbCon.createMarshaller();
                  m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
                  m.setProperty(Marshaller.JAXB_ENCODING, "ISO-8859-1");
                  File file = File.createTempFile("Fallereignis", ".xml");
                  m.marshal(fe, file);
            } catch (JAXBException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            return fe;
      }
```

Abbildung 49 Methode zum Schreiben des Fallereignisses

Für das Schreiben eines Fallereignisses muss die *Fallereignis*-Klasse mit folgenden Annotationen, in Abbildung 50 dargestellt, erweitert werden. Die *@XmlAccessorType*-Annotation definiert welche Variablen bei der Abbildung nach XML beachtet werden. Das Wurzelelement einer XML-Datei wird bei der Abbildung nach XML mit der Annotation *@XmlRootElement* gesetzt. Die Id des Fallereignisses wird mit der *@XmlAttribute*-Annotation als Attribut nach XML abgebildet. Zur Abbildung des Ereignistyps und Behandlungsfalls wird die *@XmlElement*-Annotation benötigt.

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = { "bf", "et" })
@XmlRootElement(name = "Fallereignis")
public class Fallereignis  {

      @XmlAttribute(required = true)
      private String id;

      @XmlElement(required = true)
      private Behandlungsfall bf;

      @XmlElement(required = true)
      private Ereignistyp et;

      public Fallereignis() {
      }
      //getter und setter...
}
```

Abbildung 50 An JAXB angepasstes Fallereignis

Zur Verarbeitung der Fallereignisliste, die aus der *bildeFallereignisse*-Methode zurückgeben werden, muss der Schreibvorgang für jedes Fallereignis neu gestartet werden. Dazu wird in der *starteSimulation*-Methode eine *Timer*-Instanz erzeugt, die für jedes Fallereignis einen neuen Timertask anlegt. Im Abstand des jeweiligen Fallereignisses wird die *schreibeXMLEreignis*-Methode aufgerufen. In der Abbildung 51 wird die *starteSimulation*-Methode dargestellt.

```
public void starteSimulation() {
            Timer timer = new Timer();
            Iterator<Fallereignis> it = bildeFallereignisse().iterator();
            while (it.hasNext()) {
                  fe = it.next();
                  timer.schedule(new TimerTask() {
                        public void run() {
                              schreibeXMLEreignis(fe);
                        }
                  }, new Date().getTime(), fe.getErTyp().getEreignisAbstand());
            }
      }
```

Abbildung 51 Methoden zur Simulation der Fallereignisse

Der Schreibprozess der Fallereignisse muss folgenden Aspekt berücksichtigen. Aufgrund der §21-Datenmenge entsteht eine sehr große Anzahl an Fallereignissen. Die FALL-Datei enthält 6707 Fälle, die jeweils eine Aufnahme und eine Entlassung definieren. Unter der Annahme, dass jeder Behandlungsfall lediglich eine OP hat, beläuft sich die Anzahl auf 20121 Fallereignisse, die als XML-Datei geschrieben werden. Im Fall, dass weitere Ereignistypen spezifiziert werden, steigt die Anzahl der Fallereignisse entsprechend. Beim Schreiben der Fallereignisse kann es vorkommen, dass mehrere Fallereignisse zur selben Zeit geschrieben werden müssen. In dem Fall ist ein Deadlock[16] hinsichtlich der Festplattenzugriffe vorprogrammiert. Für den Festplattenzugriff wird eine *Unmarshaller*-Instanz verwendet, die von dem *JAXBContext* zur Verfügung gestellt wird. Die JAXB-Spezifikation schreibt vor, dass die Nutzung von *JAXBContext* threadsicher sein muss. Demnach muss der Schreibprozess für ein Fallereignis als Thread definiert werden. Dazu wird diejenige Klasse, in der der Schreibprozess definiert wird, über die Schnittstelle *Runable* implementiert. Alternativ kann diejenige Klasse über die Klasse *Thread* erweitert werden, weil sie selbst die *Runable*-Schnittstelle implementiert. Zur Beseitigung des Deadlocks müssen die Schreibprozesse synchronisiert werden. Im Skript der Systemprogrammierung, siehe in (Krayl - Systemprogrammierung), werden bezüglich der Zugriffe auf globale Objekte mehrere Möglichkeiten aufgeführt. Ein Monitor ist beispielsweise ein Hilfsmittel, welches den Zugriff auf globale Objekte, in dem Fall Zugriff auf die Festplatte, synchronisiert. In der Abbildung 51 dargestellte Methode muss hinsichtlich der Synchronisierung der Schreibprozesse um eine Monitor-Datenstruktur erweitert werden.

[16] Eine Verklemmung entsteht, wenn parallele Prozesse Betriebsmittel anfordern. Die Schreibprozesse der Fallereignisse können aufgrund ihrer Abstände hinsichtlich des Festplattenzugriffs zur Verklemmung führen.

7.1.4.Webservice

Im Hinblick auf Webservices wurden die Funktionalitäten des Generators in Kapitel 4.1 als Geschäftsprozess betrachtet. Der Datenimport baut auf die Konvertierung der CSV-Daten nach XML auf. In der Konfiguration werden die Fallereignisse anhand der XML-Daten konstruiert. Die Simulation schreibt die Fallereignisse nach jeweiligem Abstand als XML-Datei. Das bedeutet dem Webservice wird als Parameter die Liste mit Fallereignissen übergeben und als Ergebnis, das jeweilige Fallereignis, als XML-Datei geschrieben. Das Schreiben der XML-Datei wird je im Abstand eines Fallereignisses durch den Simulationsservice aufgerufen. Der Simulationsservice wird durch den *Timer* kontrolliert, siehe Kapitel 7.1.3. Das zugehörige *ISimulationService*-Interface definiert zur Simulation der Fallereignisse die benötigten Methoden, siehe Kapitel 6.3 und kann mit Annotationen des Jax-WS zum Webservice deklariert werden. Das als Webservice konfiguriertes *ISimulationsService*-Interface wird in der Abbildung 52 dargestellt. Die im *ISimulationService*-Interface definierten Methoden wurden bereits in 7.1.3 implementierungsspezifisch aufgeführt.

```
@WebService
public interface ISimulationService {

	/**
	 * Methode konstruiert Fallereignisse und gibt sie als Liste zurück.
	 *
	 * @return
	 */
	@WebMethod(operationName = "bildeFallereignisse")
	@WebResult(name = "fallereignis")
	public List<Fallereignis> bildeFallereignisse();

	/**
	 * Methode schreibt schreibt das Fallereignis als XML-Datei.
	 *
	 * @param feList
	 */
	@WebMethod(operationName = "schreibeXMLEreignis")
	@WebResult(name = "fallereignis")
	public Fallereignis schreibeXMLEreignis(
			@WebParam(name = "fe") Fallereignis fe);

	/**
	 * Startet die Abarbeitung der Fallereignisse mit Timer.
	 */
	@WebMethod(operationName="starteSimulation")
	public void starteSimulation();

}
```

Abbildung 52 ISimulationService-Interface als WebService

Das *ISimulationService*-Interface bildet den Geschäftsprozess aus Kapitel 4.1 nicht vollständig ab. Es muss hinsichtlich weiterer Funktionalitäten des Generators wie

beispielsweise die Filterung der Fallereignisse anhand der Parameter der §21-Daten, erweitert werden, bzw. es müssen weitere Interfaces definiert werden, die den Geschäftsprozess vollständig abbilden.

7.2. Testarten

Zur Sicherung der Qualität in der Softwareentwicklung werden Tests durchgeführt. Je nach den Testverfahren können verschiedene Aspekte getestet werden. Das Testen einzelner Objekte oder Units ist in der Softwareentwicklung als Komponententest bekannt und kann durch den Einsatz von JUnit[17] definiert werden. Zum Testen mehrerer Objekte in einem Workflow werden Integrationstests definiert. Des Weiteren wird die Menge an Testverfahren durch die Teststufen – System- und Abnahmetest – vervollständigt.

7.2.1. JUnit-Test

Die JUnit-Tests ermöglichen das Testen einzelner Klassen oder Methoden. Im Zusammenhang mit Spring werden die JUnit-Tests mit der Annotation *@RunWith(JUnit4ClassRunner.class)* konfiguriert. Das Generator-Projekt ist hinsichtlich des Spring-Einsatzes nicht auf dem Stand, dass das JUnit-Framework eingebunden werden kann. Stattdessen werden vereinzelt Testklassen definiert, die mit JUnit getestet werden. In der Abbildung 53 wird eine Übersicht des DB4O-Datenmanagement-Projektes dargestellt, in dem in den **.test*-Packages die Testklassen definiert werden.

[17] JUnit: Ein Framework zum Testen von Klassen oder Methoden in Java.

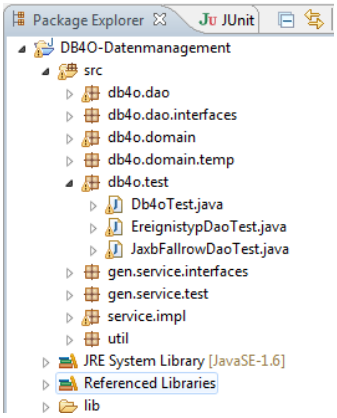


Abbildung 53 Db4o-Datenmanagement – Projektübericht

In der Abbildung 54 wird ein Ausschnitt aus der *UtilTest*-Klasse dargestellt, in dem die *parseStringToDate*- und *berechneZeitspanne*-Methoden getestet werden.

```
@Before
public void setUp() throws Exception {
	util = new Utility();
}

@After
public void tearDown() throws Exception {
	util = null;
}

@Test
public void testParseStringToDate(){
	String datum1 = "200501010001";
	String datum2 = "200501010001";
	assertNotSame(util.parseStringToDate(datum1), util.parseStringToDate(datum2));
	assertEquals(util.parseStringToDate(datum1), util.parseStringToDate(datum2));
}

@Test
public void testBerechneZeitspanne(){
	String datum1 = "200501010001";
	String datum2 = "200501010001";
	String datum3 = "201101010001";
	String datum4 = "201101040001";
	assertEquals(util.berechneZeitspanneInTagen(datum1, datum2),0);
	assertNotNull(util.berechneZeitspanneInTagen(datum1, datum2));
	assertEquals(3, util.berechneZeitspanneInTagen(datum3, datum4));
}
```

Abbildung 54 JUnitTest für Utility-Klasse

Analog können die Methoden der jeweiligen *JaxbDao*-Klassen getestet werden wie beispielsweise der *JaxbFallrowDao*-Klasse, die Methoden zum Auslesen der XML-Daten und Speichern der *Fallrow*-Objekte in der Db4o-Datenbank definiert, siehe Abbildung 55.

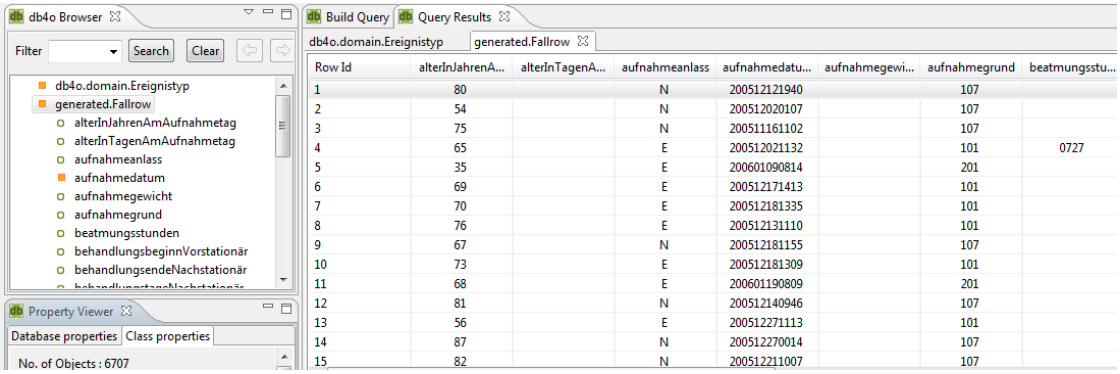
```
@Test
public void testSpeichereInDb4o() {
	Iterator<Fallrow> iterator = jaxDao
			.leseXMLObj(
					new File("C:\\Temp\\FALL.xml"),
					new File("C:\\Temp\\FALL.xsd"))
			.iterator();
	try {
		new File("db4oTest.db").delete();

		while (iterator.hasNext()) {
			fallrow = iterator.next();
			Db4oDao.store(fallrow);
			Db4oDao.commit();
		}
	} finally {
		Db4oDao.close();
	}
	assertNotNull(Db4oDao.query(Fallrow.class));
	List<Fallrow> fallList = Db4oDao.query(Fallrow.class);
	assertEquals(6707, fallList.size());
}
```

Abbildung 55 Test auf Persistenz der Fallzeilen der FALL.xml in Db4o

An der Stelle ist zu erwähnen, dass der Test hinsichtlich der Persistenz ungefähr 22 Minuten dauert, weil alle 6707 Fallzeilen in der db4o-Datenbank persistiert werden. Die Anzahl der Zeilen in db4o-Datenbank muss mit der Anzahl der Zeilen in der CSV- oder XML-Datei übereinstimmen.

7.2.2.Integrationstest

Hinsichtlich der Funktionalitäten können folgende Aussagen gemacht werden. Das Konvertieren der CSV-Daten nach XML sowie das Auslesen der XML-Daten funktioniert. Beim Einlesen der XML-Daten wird das zugehörige XML-Schema zur Validierung verwendet. An der Stelle ist zu erwähnen, dass das XML-Schema aus den XML-Daten abgeleitet wird und somit lediglich die Datentypen validiert werden. Die Funktionalität, Auslesen der XML-Daten, wird in der *JaxbFallrowDao*-Klasse eingesetzt. Hinsichtlich der Ereignistypen – Aufnahme, OP und Entlassung – werden zunächst die FALL- und OPS-XML-Dateien herangezogen. Die einzelnen Zeilen werden in der db4o-Datenbank persistiert. Mit dem Object-Manager-Enterprise-Plugin (OME) können die persistierten Objekte der db4o-Datenbank tabellarisch dargestellt werden. In der Abbildung 56 wird ein Ausschnitt des OME bezüglich der *Fallrow*-Objekte dargestellt.



Row Id	alterInJahrenA...	alterInTagenA...	aufnahmeanlass	aufnahmedatu...	aufnahmegewi...	aufnahmegrund	beatmungsstu...
1	80		N	200512121940		107	
2	54		N	200512020107		107	
3	75		N	200511161102		107	
4	65		E	200512021132		101	0727
5	35		E	200601090814		201	
6	69		E	200512171413		101	
7	70		E	200512181335		101	
8	76		E	200512131110		101	
9	67		N	200512181155		107	
10	73		E	200512181309		101	
11	68		E	200601190809		201	
12	81		N	200512140946		107	
13	56		E	200512271113		101	
14	87		N	200512270014		107	
15	82		N	200512211007		107	

Abbildung 56 Darstellung der Fallrows in db4o mit OME

Weitere Funktionalitäten, wie das Bilden der Behandlungsfälle, Konstruieren der Fallereignisse und Simulation der Fallereignisse, können aufgrund des aktuellen Projektzustandes nicht getestet werden.

7.2.3.Plausibilitätstest

Zur Plausibilität der §21-Daten wurden in Kapitel 5.4 zwei Aspekte – Format- und Datenprüfung – definiert.

Zur Formatprüfung gehört unter Anderem die Prüfung auf Anzahl der Datenfelder. In dem Fall wird auf den Konvertierungsprozess zurückgegriffen, der die geparsten CSV-Daten nach XML überführt. In der Abbildung 57 wird beispielsweise die Anzahl der Datenfelder in der Datei FALL.csv anhand eines JUnit-Tests überprüft.

```
@Test
public void testAnzahlDatenfelder() throws ParseException{
	String[] header = null;
	header = csvt.nextLine();
	assertNotNull(header);
	assertEquals(26, header.length);
}
```

Abbildung 57 Prüfung auf Anzahl der Datenfelder

Die *CSVTokenizer*-Klasse stellt eine *nextLine*-Methode zur Verfügung, die ein Array mit den Spaltennamen zurückgibt. In der FALL-Datei werden 26 Datenfelder erwartet.

Das Ergebnis der Array-Länge muss dem erwarteten Ergebnis entsprechen, damit der Test erfolgreich abgeschlossen wird. Analog können die restlichen CSV-Daten auf Anzahl der Datenfelder überprüft werden.

Die Prüfung auf Redundanz der Fallnummern, damit ist das krankenhausinterne Kennzeichen gemeint, kann implementierungsspezifisch mit folgenden Datenstrukturen realisiert werden. Die *Fallrow*-Klasse der §21-Domäne führt eindeutige Fälle, das bedeutet, dass jede Zeile der FALL-Datei hinsichtlich des krankenhausinternen Kennzeichens nur einmal vorkommen darf. In dem Fall muss die *Fallrow*-Klasse die *equals*- und *hashCode*-Methode überschreiben. In der *equals*-Methode wird das krankenhausinterne Kennzeichen als Vergleichskriterium definiert. Die *equals*-Methode liefert *true* zurück, wenn zwei *Fallrow*-Objekte gleich sind, wobei die *hashCode*-Methode ebenfalls das gleiche Ergebnis liefern muss, siehe in (Esser 2008 S.45). In einer Datenstruktur wie *List*, siehe Java-API, können die *Fallrows* anhand des krankenhausinternen Kennzeichens auf Redundanz analysiert werden.

Zur Prüfung der Fallnummern auf Fehler kann der Zeichensatz der Fallnummer lediglich auf fremde Zeichen, in dem Fall alles außer Zahlen, analysiert werden. Zur genauen Prüfung müssen die Konventionen bekannt sein, nach denen eine Fallnummer als richtig anzusehen ist.

Die Datenprüfung bezieht sich auf den Inhalt der §21-Daten, das bedeutet die Werte der Datenfelder müssen analysiert und bestimmte Konstellationen wie beispielsweise eine Entlassung vor einer Aufnahme auf Plausibilität überprüft werden. Wenn bei einem Behandlungsfall das Entlassungsdatum vor seinem Aufnahmedatum liegt, dann sind die §21-Daten des entsprechenden Behandlungsfalls nicht korrekt. Für solche Prüfungen müssen die entsprechenden Datenfelder bezüglich des Datums verglichen werden. Die Date-Klasse, siehe Java-API, stellt dafür die *before*- und *after*-Methode zur Verfügung, anhand der ein Aufnahme- oder Entlassungsdatum auf zeitliche Konsistenz der Ereigniskette, siehe Kapitel 5.1.1.4, überprüft werden kann.

Des Weiteren muss hinsichtlich der 1:n-Beziehung der §21-Domäne, siehe Kapitel 5.4, geprüft werden, ob zu jedem *Fallrow*-Objekt passende *Fabrow*-, *Opsrow*-, *Icdrow*- oder *Entgelterow*-Objekte existieren. Die Referenzierung erfolgt über das krankenhausinterne Kennzeichen. Die Objekte stehen in *Collections*, siehe Java-API, zur Verfügung. Das bedeutet das *Fallrow*-Objekt wird beispielsweise mit dem Opsrow-Objekt der Opsrow-Objektmenge verglichen und bei gleichem krankenhausinternem Kennzeichen zum *Behandlungsfall*-Objekt hinzugefügt. Die Java-API bietet zum Vergleichen von Objekten zwei Möglichkeiten. Zum einen das

Comparable-Interface, welches über die *compareTo*-Methode ein Objekt vergleichbar macht. In dem Fall müssen die Klassen wie beispielsweise die *Fallrow*- und Opsrow-Klasse das *Comparable*-Interface implementieren und in der *compareTo*-Methode wird das krankenhausinterne Kennzeichen als Kriterium definiert, nach dem sich das *Fallrow*-Objekt unterscheidet. Zum Anderen wird von der Java-API das *Comparator*-Interface zur Verfügung gestellt, welches mit der *compare*-Methode zwei Objekte miteinander vergleicht. Eine Klasse, die zwei Objekte über die *compare*-Methode vergleicht, wird als *Comparator* bezeichnet. Beide Implementierungsansätze definieren eine Ordnung, anhand der die Objekte verglichen werden können, wobei der *Comparator* im Gegensatz zum *Comparable* mehrere Definitionen bezüglich der Ordnungen ermöglicht und somit mehrere Vergleichskriterien definiert werden können. Näheres zum Comparator und Comparable kann in (Galileo Computing - Java ist auch eine Insel) nachgelesen werden.

Zur Prüfung weiterer Konstellationen wie beispielsweise, Schwangerschaft bei einem männlichen Behandlungsfall, müssen entsprechende Datenfelder herangezogen werden. In der Vereinbarung der *InEK* werden alle Datenfelder der §21-Daten erläutert, sie bieten die Basis zur Überprüfung weiterer möglicher Konstellationen, die an der Stelle nicht behandelt werden.

8. Ergebnisse

In diesem Kapitel werden die gesammelten Erfahrungen mit dem Generator-Projekt sowie der aktuelle Stand des Projektes präsentiert.

Das Generator-Projekt hat sich rückblickend als ein sehr umfangreiches Projekt erwiesen. Es berücksichtigt die Technologieanalyse, rechtliche Aspekte im Gesundheitssystem sowie die Konzeption und Entwicklung des Generators. Der Generator sollte dazu beitragen, die Population im virtuellen Gesundheitssystem nachzubilden. Die Ergebnisse des Projektverlaufs werden in folgende Teilbereiche eingeteilt.

- Technologieanalyse: Zur Realisierung des Generator-Projektes wurden zahlreiche Technologien, siehe Kapitel 3, in Betracht gezogen. Die Analyse hat ergeben, dass die Technologien zwar ihren Zweck im Generator-Projekt hinsichtlich der Implementierung erfüllen können, ihr Einsatz jedoch viel Detailwissen erfordert.
- Konzept: Die Ergebnisse der Anforderungsanalyse wurden in Kapitel 5 zusammengetragen. Der architektonische Ansatz hinsichtlich der Komponenten wird in Kapitel 6 beschrieben. Beide Aspekte sowie der GUI-Prototyp werden im Pflichtenheft gesondert aufgeführt.
- Implementierung: Aufgrund zahlreicher Implementierungsprobleme konnte das Generator-Projekt nicht vollständig abgeschlossen werden. Zum aktuellen Zeitpunkt liegen folgende Funktionalitäten vor, die in Kapitel 7 prototypspezifisch implementiert wurden.
    - Die Konvertierung der CSV-Daten nach XML und deren Validierung gegen das zugehörige XML-Schema.
    - Die XML-Daten können zur temporären Weiterverarbeitung wie beispielsweise Bildung der Fallereignisse in eine objektorientierte Datenbank gebracht werden.
    - Die Ereignistypen können verwaltet werden.
    - Es liegt ein Ansatz zur Berechnung des Vorgängers und Abstandes für ein Fallereignis vor. Des Weiteren liegt ein Ansatz zur Simulation der Fallereignisse vor. Aufgrund der Komplexität der SODA-Anfragen auf die db4o-Datenbank können jedoch diese Ansätze nicht getestet werden.

- Aufgrund der Priorität der Zeitkomponente wurde auf die Entwicklung des Benutzerinterfaces verzichtet.

9. Diskussion und Ausblick

In diesem Kapitel werden das Generator-Projekt sowie dessen Probleme und Lösungsansätze reflektiert. Des Weiteren werden mögliche Erweiterungen vorgestellt.

In den Kapiteln 7 und 8 wurde bereits angedeutet, dass die Realisierung nicht fertiggestellt werden konnte. Zum Einen lag es daran, dass die Auswahl der Technologien nicht optimal war. Nicht optimal in dem Sinne, dass zu Projektbeginn noch keine Erfahrungen mit den Technologien vorhanden waren. Zum Anderen erschwerte die §21-Datenstruktur die Analyse der Domänenobjekte. Auf den ersten Blick beschreibt die §21-Domäne eine einfache 1:n-Beziehung der Datei FALL zu den FAB-, OPS-, ICD- und ENTGELTE-Dateien, siehe Kapitel 5.4. Zur Spezifikation der Ereignistypen müssen sämtliche Datenfelder der aufgeführten Dateien analysiert werden, die im Gesamtbild ihre Komplexität entfalten. Des Weiteren erschwert die Dynamik im DRG-Fallpauschalensystem die Aufstellung eines beständigen Domänenmodells der §21-Daten. Diese Dynamik ist bedingt durch eine kontinuierliche Weiterentwicklung des DRG-Systems Jedes Jahr kommen neue DRGs hinzu, beispielsweise aufgrund von Gesetzesänderungen oder neuer Krankheitsfälle, die Änderungen bei den §21-Daten hervorrufen können. Dieser Sachverhalt spiegelt sich zumeist bei den Bezeichnungen der Datenfelder der §21-Daten wider. Sie können in der Bezeichnung oder deren Anzahl variieren. Unabhängig davon wie die §21-Daten im Backend gehalten werden, also ob relational, xml- oder objektorientiert, ziehen Änderungen in den §21-Daten Änderungen am Domänenmodell bzw. an den Datenfeldern der einzelnen Klassen nach sich. An der Stelle ist zu erwähnen, dass je nach dem welche Datenfelder der §21-Daten von Jahr zu Jahr geändert werden, jährlich eine Anpassung des Domänenmodells stattfinden muss.

Zur Berücksichtigung der Änderungen der §21-Daten und weiterer Aspekte wie beispielsweise Interoperabilität, Webservice und das XML-Output-Format des Generators, fiel die Entscheidung die CSV-Daten zu Beginn des Datenimportes nach XML zu überführen sowie JAXB zum Parsen und Serialisieren der XML-Daten einzusetzen. Die Idee geht zum Teil auf, es ist in der Tat möglich, §21-XML-Daten mit dem JAXB-Einsatz zu parsen oder wieder zu serialisieren. Jedoch entsteht ein Problem durch die 1:n-Beziehung der §21-Daten, die als Behandlungsfälle gekapselt werden. Zur Konstruktion der Fallereignisse müssen zur Laufzeit neben den Ereignistypen die Behandlungsfälle zur Verfügung stehen, dies ist aber aufgrund des

Unmarshallers von JAXB nicht möglich, siehe Kapitel 7.1.2. An dieser Stelle muss auf den *UnmarshallHandler* mit Einsatz des SAX-Parsers ausgewichen werden. Alternativ – wenn man bei der XML-Verarbeitung mit JAXB bleibt – muss die 1:n-Beziehung der §21-Daten XML-spezifisch, also die XML-Daten und das zugehörige XML-Schema, angepasst werden. Konkret bedeutet es, dass die nach XML konvertierten §21-Daten als XML-Datei zentralisiert werden. In der zugehörigen XML-Schema-Datei wird die 1:n-Beziehung der §21-Daten vollständig erfasst. Als Ergebnis liegt eine XML-Datei, die alle XML-Daten aus FALL-, FAB-, OPS-, ICD- und ENTGELTE-Dateien enthält und eine XML-Schema-Datei vor. Beide Ansätze erfordern weiteres Detailwissen der JAXB-Technologie, welches im Projektverlauf aus Zeitgründen nicht zur Verfügung stand.

Die Kapselung der §21-Daten zu einem Behandlungsfall war zur Laufzeit mit JAXB allein ohne Weiteres nicht möglich, deshalb wurde eine Datenbank in Erwägung gezogen. Das Parsen der XML-Daten und wieder Serialisieren bedeutet, dass lediglich mit XML und Java-Objekten gearbeitet wird. Aus diesem Grund wurde db4o als objektorientiertes Backend ausgewählt. Im Zusammenhang mit Spring sollte JDO mit Datanucleus die Datenzugriffsschicht auf db4o abbilden. Die dabei entstandene Problematik, also die Nichtunterstützung der *persist*-Methode zum Speichern von Objekten in der db4o-Datenbank, wurde bereits in Kapitel 6.2 beschrieben. Die Definition der Db4oDao-Klasse, siehe Kapitel 7.1.2, ermöglichte die Verwaltung der zu persistierenden Objekte.

Zur Konstruktion der Fallereignisse sollten die Behandlungsfälle mit den Ereignistypen zur Laufzeit verknüpft werden. Für die Verknüpfung gelten Bedingungen, die sich je nach Ereignistyp aus den Datenfeldern der §21-Daten ergeben. Sowohl die Behandlungsfälle als auch die Ereignistypen werden bei der Anfrage auf db4o in *Collections*, siehe Java-API, zurückgegeben. Jedes *Fallrow*-Objekt hat aufgrund der 1:n-Beziehung weitere Mengen wie beispielsweise *Opsrows*. Zum Abgreifen bestimmter Datenfelder in diesen Mengen müssen spezielle SODA-Queries gebildet werden. Des Weiteren müssen die Anfrageergebnisse wiederum in Mengen zur Konstruktion von Fallereignissen weiter verarbeitet werden. An der Stelle ist zu erwähnen, dass hierfür noch wenig Detailwissen über Anfragen in db4o vorliegt, so dass zwar ein Ansatz zu Kapselung von Behandlungsfällen implementierungsspezifisch vorliegt, siehe die *BehandlungfallDao*-Klasse im DB4O-Datenmanagement-Projekt, der aber nicht zum gewünschten Ergebnis geführt hat.

Der Geschäftsprozess des Generators definiert neben dem Datenimport ebenfalls die Konfigurationskomponente, die als Benutzerinterface dem Benutzer zur Verfügung gestellt werden soll. Der Ansatz des GUI-Mockups im Pflichtenheft definiert mehrere Aspekte, die allein mit dem Einsatz von Spring-MVC nicht möglich sind. Es geht dabei beispielsweise um die Validierung von Benutzereingaben, dynamisches Füllen von Listen oder den Fortschrittsbalken für einen andauernden Prozess. Diese Aspekte können erst mit Einsatz von Ajax berücksichtigt werden. In dem Fall muss das DWR-Framework, siehe Kapitel 3.5.4, eingesetzt werden. Folglich könnte man die Frage stellen, warum nicht das GWT-Framework ausgewählt wurde. Bei der Konzeption und Entwicklung lag die Priorität zum Einen auf Webservice und zum Anderen auf Interoperabilität. Die Analyse hinsichtlich der Technologien ergab, dass das GWT-Framework lediglich den RPC-Mechanismus zulässt, was nicht mit SOAP, das nachrichtenbasiert über XML kommuniziert, gleichzusetzen ist. GWT war daher für den vorgesehenen Einsatzzweck nicht ausreichend. Die Webservices über SOAP sind XML-spezifisch und XML ermöglicht die Interoperabilität. Aus diesen Gründen wurde der Ansatz verfolgt, die CSV-Daten sofort beim Datenimport nach XML zu konvertieren. JAXB sollte dabei nicht nur beim Webservice, sondern auch beim Auslesen der XML-Daten eine Rolle spielen.

In Kapitel 8 wurde bereits aufgeführt, dass das Generator-Projekt nicht abgeschlossen werden konnte. An dieser Stelle werden Tätigkeiten aufgeführt, die im Fall der Projektfortführung relevant sein können.

- Die 1:n-Beziehung der §21-Daten sollte vollständig als XML-Schema abgebildet werden. Jede weitere Änderung der §21-Datenfelder kann in das XML-Schema eingearbeitet werden und somit die Aktualisierung des Domänenmodells vereinfachen. Das lernende Prinzip des DRG-Systems, welches jedes Jahr neue DRGs definiert, sollte auf die Evolution des XML-Schemas, welches die Änderungen in den §21-Daten berücksichtigt, übertragen werden.
- Die Kapselung der §21-Daten zu den Behandlungsfällen ist für den Einsatz von *UnmarshallHandler* und SAX-Parser zu untersuchen. Im Erfolgsfall können bereits beim Einlesen der §21-XML-Daten die Behandlungsfälle spezifiziert werden. Das erspart vorheriges Persistieren der einzelnen Zeilentypen[18] in der db4o-Datenbank.

[18] Die Zeilentypen werden durch die Klassen Fallrow, Fabrow, Opsrow, Icdrow und Entgelterow definiert.

- Zur Entwicklung des Ajax-fähigen Benutzerinterface sollte das DWR-Framework analysiert werden.
- Des Weiteren muss das Interface, welches die Funktionalitäten als Webservice zur Verfügung stellen soll, auf Granularität der Abbildung von Funktionalitäten analysiert werden. Es ist zu analysieren welche Funktionalität auf dem Client zu verrichten ist und welche als Webservice zur Verfügung gestellt werden kann. Denn es hat einen großen Einfluss auf die Performanz, ob die Liste mit Fallereignissen clientseitig konstruiert und serverseitig abgearbeitet wird oder ob die §21-Daten zunächst auf den Server geladen werden und jede Tätigkeit zur Verarbeitung der §21-Daten, also die Kapselung der §21-Daten zu den Behandlungsfällen oder die Konstruktion der Fallereignisse, netzwerktechnisch Traffic verursacht.

Unabhängig vom Stand des Generator-Projektes können im Hinblick auf Erweiterungen des Generators folgende Überlegungen hilfreich sein.

Die §21-Daten erfordern den Zugriff auf den ICD- und OPS-Katalog des DIMDI. In Anbetracht der Tatsache, dass jedes Jahr die Kataloge weiterentwickelt werden, besteht der Grund zur Annahme, dass die Bedeutung der ICD- und OPS-Codes sowie der DRGs jährlich variieren kann. Generell muss zur Prüfung der Plausibilität unter Anderem die Konsistenz der ICD- und OPS-Codes sowie der Entgeltarten gegeben sein. Dazu muss die Möglichkeit gegeben sein, je nach Datenjahr der §21-Daten, auf entsprechende ICD- OPS- und DRG-Kataloge zuzugreifen. Auf diese Weise können Inkonsistenzen bei den abrechnungsrelevanten Datenfeldern aufgedeckt werden. Des Weiteren können Fallereignisse unabhängig von einem Datenjahr konstruiert und simuliert werden. Die Integration der ICD- und OPS-Kataloge des *DIMDI* könnte zum aktuellen Zeitpunkt auf folgendem Weg geschehen. Die entsprechenden Kataloge werden unter Anderem als Textdateien zur Verfügung gestellt, die von der *DIMDI*-Webseite heruntergeladen werden können. Übertragen auf das Generator-Projekt müsste die Komponente Datenmanagement um einen Parser erweitert werden, der die entsprechenden Textdateien einliest und zur Laufzeit die Daten des ICD- und OPS-Katalogs zur Verifizierung der §21-Daten zur Verfügung stellt. Auch die Integration des DRG-Systems wäre denkbar, dazu stellt *InEK* den sogenannten DRG-Browser zur Verfügung, der in Form einer Access-Datenbank vorliegt. In dem Fall muss der Generator eine Schnittstelle definieren, die den Zugriff auf die Access-Datenbank des DRG-Browsers ermöglicht.

Eine weitere Überlegung, die aus der rechtlichen Sicht einen Einfluss auf den Generator haben könnte, ist die Berücksichtigung des §17d des Krankenhausfinanzierungsgesetzes. Er definiert ein durchgängig leistungsorientiertes, pauschalierendes Vergütungssystem mit tagesbezogenen Entgelten, das ab dem 01.01.2013 eingeführt werden soll (DKR-Psych 2010, *Inek* GmbH). Die Ergänzung „tagesbezogene Entgelte“ definiert das Entgeltsystem neu, welches die relevanten Informationen zum Krankheits- und Leistungsspektrum in psychiatrischen und psychosomatischen Einrichtungen berücksichtigen muss. Zu den relevanten Informationen gehören vor allem die zeitintensiven Leistungen, die in den psychiatrischen Einrichtungen entstehen, die jedoch anhand der zur Verfügung stehender OPS-Prozeduren nicht exakt abgerechnet werden können. Folglich müssen zur Abrechnung der Leistungen zusätzliche Informationen wie beispielsweise taggenaue Abstufung der Patienten nach der PsychPV-Kategorie übermittelt werden (Hauth 2010). Die Psychiatrie-Personalverordnung (PsychPV) regelt die Personalausstattung der in den stationären oder teilstationären Behandlungsprozess einbezogenen Berufsgruppen. Für verschiedene Behandlungsbereiche werden Zeitwerte in Minuten je Patient und Woche erfasst, die dann von den Personalstellen umgerechnet werden (f&w Haas, Leber 2010). Demnach bewirkt der Paradigmenwechsel im Entgeltsystem möglicherweise eine Erweiterung der OPS-Prozeduren, damit vor allem zeitintensive Leistungen abgebildet werden können. Dies hat wiederum Einfluss auf das Spektrum der Ereignistypen, die zur Simulation der Fallereignisse definiert werden und sich nicht nur in der Anzahl sondern auch in der Komplexität verändern werden.

Trotz der sich erst später herausstellenden Komplexität konnten wichtige Teile des Generator-Projektes umgesetzt werden. Für andere Teile wurden Lösungsgansätze gefunden und genannt, dass eine solide Grundlage zu weiteren Entwicklungen des Generator-Projektes geschaffen wurde.

Externe Dokumente

Zu den externen Dokumenten gehören die Dokumente zur Datenlieferung, das Lastenheft und das Pflichtenheft. Die Dokumente zur Datenlieferung spezifizieren die §21-Datenstruktur und gehören zu den wichtigen Arbeitsmitteln hinsichtlich der Spezifikation weiterer Ereignistypen. Das Pflichtenheft baut auf dem Lastenheft auf und beschreibt die Anforderungen, die an den Generator gestellt wurden sowie die eingesetzte Entwicklungsumgebung. Die Dokumente werden im Anhang auf einer CD mitgeliefert. Die Dokumente zur Datenlieferung können auch auf der InEK-Homepage (Datenlieferung gem. §21 KHEntgG, InEK GmbH) heruntergeladen werden.

Das Projekt ist noch nicht abgeschlossen, folglich können hinsichtlich der Anforderungen an den Generator neue Ideen oder Erweiterungen in das Pflichtenheft aufgenommen werden.

Abkürzungsverzeichnis

Ajax:

Asynchronous Java Script and XML

ACID:

Atomicity Consistency Isolation Durability

API:

Application Programming Interface

CRUD:

Create Read Update Delete

CSS:

Cascading Style Sheet

CSV:

Comma Separated Value

DAO:

Data Access Object

db4o:

objektorientierte Datenbank

DICOM:

Digital Imaging and Communications in Medicine

DIMDI:

Deutsches Institut für Medizinische Dokumentation und Information

DOM:

Document Object Model

DRG:

Diagnostic Related Groups

DTO:

Data Transfer Object

DWR:

Direct Web Remoting

EDIFACT:

Electronic Data Interchange For Administration, Commerce and Transport

EL:

Expression Language

FTP:

File Transfer Protocol

GKV:

Gesetzliche Krankenversicherung

GWT:

Google Webtoolkit

HL7:

Health Level 7

HTML:

Hyper Text Markup Language

HTTP:

Hypertext Transfer Protocol

IHE:

Integrating the Healthcare Enterprise

InEK:

Das Institut für Entgelte im Krankenhaus

IP:

Internet Protocol

IQWiG:

Institut für Qualität und Wirtschaftlichkeit im Gesundheitswesen

Java-EE:

Java Platform Enterprise Edition

JAXB:

Java Architecture for XML Binding

Jax-WS:

Ein Framework zur Entwicklung von Webservices

JDBC:

Java Database Connectivity

JDO:

Java Data Objects

JDOQL:

Java Data Objects Query Language

JSF:

Java Server Faces

JSON:

Java Script Object Notation

JSP:

Java Server Pages

JPA:

Java Persistence API

JSTL:

Java Server Pages Standard Library

KHEntgG:

Krankenhausentgeltgesetz

KHG:

Krankenhausfinanzierungsgesetz

KIS:

Krankenhausinformationssystem

LDAP:

Lightweight Directory Access Protocol

MVC:

Model View Controller

ORM:

Object-Relational Mapping

OSI-Referenzmodell:

Open Systems Interconnection Reference Model

OXM:

Object-XML Mapping

POJO:

Plain Old Java Object

Psych-PV:

Psychiatrie-Personalverordnung

QBE:

Query by Example

REST:

Representational State Transfer

RMI:

Remote Method Invocation

RPC:

Remote Procedure Call

SGBV:

Sozialgesetz fünftes Buch

SOA:

Serviceorientierte Architektur

SOAP:

Simple Object Access Protocol

SODA:

Simple Object Data Access

STS:

Source ToolSuite von Spring

TCP:

Transmission Control Protocol

UML:

Unified Modeling Language

URI:

Uniform Resource Identifier

WADL:

Web Application Description Language

WSDL:

Web Services Description Language

WWWC:

World Wide Web Consortium

XML:

Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Literaturverzeichnis

Internetquellen

8 Möglichkeiten, Projekte mit Spring aufzubauen. Verfügbar unter: http://it-republik.de/jaxenter/news/8-Moeglichkeiten-Projekte-mit-Spring-aufzubauen-054379.html [Zugegriffen März 11, 2011].

DataNucleus Access Platform - JDO Tutorial. Verfügbar unter: http://www.datanucleus.org/products/accessplatform/guides/jdo/tutorial.html [Zugegriffen März 11, 2011].

Datenlieferung gem. § 21 KHEntgG, InEK GmbH. Verfügbar unter: http://www.g-drg.de/cms/index.php/Datenlieferung_gem._21_KHEntgG [Zugegriffen Dezember 29, 2010].

Datenstelle, InEK GmbH. Verfügbar unter: http://www.g-drg.de/cms/index.php/inek_site_de/Datenlieferung_gem._21_KHEntgG/Datenstelle [Zugegriffen Dezember 29, 2010].

db4o Reference Documentation. Verfügbar unter: http://developer.db4o.com/Documentation/Reference/db4o-8.0/java/reference/ [Zugegriffen März 11, 2011].

DIMDI - Thesauri. Verfügbar unter: http://www.dimdi.de/static/de/klassi/mesh_umls/index.htm [Zugegriffen März 12, 2011].

DKR-Psych 2010, InEK GmbH. Verfügbar unter: http://www.g-drg.de/cms/index.php/Psychiatrie_Psychosomatik/Kodierrichtlinien/DKR-Psych_2010 [Zugegriffen März 16, 2011].

f&w Haas Leber 2010. Verfügbar unter: http://www.gkv-spitzenverband.de/upload/f&w_2010_01_Haas_Leber_Neue_Psych-Entgelte_13741.pdf [Zugegriffen März 14, 2011].

Galileo Computing Java ist auch eine Insel (8. Auflage) – 12.4 Vergleichen von Objekten. Available at: http://www.iks.hs-merseburg.de/~uschroet/Literatur/Java_Lit/JAVA_Insel/javainsel_12_004.htm [Zugegriffen März 11, 2011].

Integrierte Versorgung 2004 Ingenerf. Verfügbar unter: http://www.imi2.uni-luebeck.de/~ingenerf/publications/Integrierte%20Versorgung%202004%20Ingenerf.pdf [Zugegriffen Dezember 29, 2010].

IQWiG – Gesetzliche Grundlage. Verfügbar unter: https://www.iqwig.de/gesetzliche-grundlage.933.html?random=21fda8 [Zugegriffen März 10, 2011].

Java Data Objects 3 - Maintenance Release 3. Verfügbar unter: http://jcp.org/aboutJava/communityprocess/mrel/jsr243/index3.html

[Zugegriffen März 11, 2011].

JAXB zur XML-Verarbeitung. Verfügbar unter: http://www.torsten-horn.de/techdocs/java-xml-jaxb.htm [Zugegriffen März 11, 2011].

Spiralmodell (PNG-Grafik, 600x493 Pixel). Verfügbar unter: http://www.mprove.de/script/05/im/_media/Abb1Spiralmodell.png [Zugegriffen März 11, 2011].

Statistisches Jahrbuch 2010. Verfügbar unter: http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/SharedContent/Oeffentlich/B3/Publikation/Jahrbuch/StatistischesJahrbuch,property=file.pdf [Zugegriffen Dezember 29, 2010].

Literaturquellen

Ahrendts, F. & Marton, A., 2007. *IT-Risikomanagement leben!: Wirkungsvolle Umsetzung für Projekte in der Softwareentwicklung*, Springer.

Bärwolff, H., Victor, F. & Hüsken, V., 2006. *Handbuch IT-Systeme in der Medizin: IT-Entscheidungshilfen für den Medizinbereich.Konzepte, Standards und optimierte Prozesse*, Vieweg +Teubner.

Bauer, G., 2008. *Architekturen für Web-Anwendungen: Eine praxisbezogene Konstruktions-Systematik*, Vieweg +Teubner.

Behrendt, J. & Zeppenfeld, K., 2008. *Web 2.0*, Springer.

Dwyer, J., 2008. *Pro Web 2.0 Application Development with GWT*, Apress.

Esser, F., 2008. *Java 6 Core Techniken: Essentielle Techniken für Java-Apps*, Oldenbourg Wissenschaftsverlag.

Franke, U., 2008. *Asset Securitization im Gesundheitswesen: Erfahrungen in den USA und anderen Ländern als Basis einer Abwägung von Einsatzmöglichkeiten in Deutschland*, DUV.

Haas, P., 2006. *Gesundheitstelematik: Grundlagen, Anwendungen, Potenziale*, Springer.

Hanson, R.D. & Tacy, A., 2007. *GWT im Einsatz: Ajax-Anwendungen entwickeln mit dem Google Web Toolkit*, Hanser Fachbuchverlag.

Harnisch, C., 2007. *Routing& Switching*, Hüthig Jehle Rehm.

Hauth, I., 2010. Was das neue Entgeltsystem für Psychiatrie und Psycosomatik mit sich bringt, Springer-Medizin.

Heuser, O. & Holubek, A., 2009. *Java Web Services in der Praxis: Realisierung einer SOA mit WSIT, Metro und Policies*, Dpunkt.Verlag GmbH.

Höhl, 2010. Patienten online werben.

Johner, C. & Haas, P., 2009. *IT im Gesundheitswesen*, Hanser.

Kleuker, S., 2008. *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*, Vieweg +Teubner.

Krayl - Systemprogrammierung, Skript zur Vorlesung Systemprogrammierung, HS-Heilbronn

Lehmann, T.M., 2004. *Handbuch der medizinischen Informatik*, Hanser Verlag.

Liebhart, D., 2007. *Architecture blueprints: ein Leitfaden zur Konstruktion von Softwaresystemen mit Java Spring, .NET, ADF, Forms und SOA*, Hanser Verlag.

Louis, D. & Müller, P., 2007. *Das Java 6 Codebook*, Pearson Education.

McLaughlin, B., Edelson, J. & Online, S.T.B., 2006. *Java & XML*, O'Reilly Media, Inc.

Michaelis, S. & Schmiesing, W., 2006. *Jaxb 2.0: Ein Programmiertutorial für die Java Architecture for XML Binding*, Hanser Verlag.

Müller, B., 2006. *JavaServer Faces: ein Arbeitsbuch für die Praxis*, Hanser Verlag.

Münz, S., 2006. *Professionelle Websites: Programmierung, Design und Administration von Webseiten*, Addison-Wesley.

Oates, R. & Bachlmayer, G., 2008. *Spring& Hibernate: Eine praxisbezogene Einführung*, Hanser Verlag.

Paterson, J. u. a., 2006. *The definitive guide to db4o*, Apress.

Pfetzing, K. & Rohde, A., 2009. *Ganzheitliches Projektmanagement*, Verlag Goetz Schmidt.

Richardson, L. & Ruby, S., 2007. *Web Services mit REST*, O'Reilly Germany.

Saake, G. & Sattler, K., 2003. *Datenbanken & Java*, Dpunkt-Verl.

Schlegel, H., 2010. *Steuerung der IT im Klinikmanagement*, Vieweg+Teubner.

Scholz, M. & Niedermeier, S., 2009. *Java und XML: Alles zu DOM, SAX, JAXP, StAX. JAXB und Webservices sowie den Grundlagen des XML-Publishing-Prozesses*, Galileo Press GmbH.

Schreiner, R., 2009. *Computernetzwerke: von den Grundlagen zur Funktion und Anwendung*, Hanser Verlag.

STEFAN, T., 2009. *REST UND HTTP: EINSATZ DER ARCHITEKTUR DES WEB FUR INTEGRATIONSSZ*, DPUNKT VERLAG.

Steyer, R., 2007. *Google Web Toolkit*, Entwickler.press.

Turau, V., Saleck, K. & Lenz, C., 2004. *Web- basierte Anwendungen entwickeln mit JSP2: Einsatz von JSTL, Struts, JSF, JDBC, JDO, JCA*, Dpunkt-Verl.

Walls, C. & Breidenbach, R., 2007. *Spring im Einsatz*, Hanser Fachbuchverlag.

Wolff, E., 2009. *Spring 3: Framework für die Java-Entwicklung*, Dpunkt.Verlag GmbH.