HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Automatische Arztbriefgenerierung

DIPLOMARBEIT

Rudolf Erich Robinigg

Studiengang Medizinische Informatik
Hochschule Heilbronn / Universität Heidelberg

Erstbetreuer: Prof. Dr.-Ing. Daniel Pfeifer
Zweitbetreuer: Prof. Dr. Christian Fegeler
Firma: innoForce Est., Liechtenstein

Bearbeitungszeitraum: 15.08.2010 – 15.02.2011
Datum der Abgabe: 15.02.2011

Inhaltsverzeichnis

Abstract

Dokumentation ist ein wichtiger Bestandteil der täglichen Arbeit im Krankenhaus. Die Anforderungen an die Dokumentation der Krankengeschichte steigen immer mehr, unter anderem durch gesetzliche Rahmenbedingungen wie Aufbewahrungspflicht oder Abrechnungsmodi. Der Zeitaufwand für die Dokumentation fehlt dem Arzt später für seine eigentliche Aufgabe, der medizinischen Betreuung von Patienten.

Durch gesteigerte Mobilität von Patienten und Spezialisierung von Ärzten findet die Behandlung oft nicht nur an einer Stelle statt. Dadurch steigen die Anforderungen an die Kommunikation der Stellen untereinander. Der Austausch der Daten wird damit zu einer zentralen Aufgabe schon während der Behandlung.

Es ist die Aufgabe der Hersteller von Software für das Gesundheitswesen, den Kliniken geeignete Tools zur Verfügung stellen, welche die gezwungenermaßen erfassten Daten dem Arzt für seine tägliche Arbeit in geeigneter Form zusammenzufassen. Herstellerübergreifend sind Absprachen notwendig, damit unterschiedliche Softwaresysteme in der Lage sind, miteinander Daten auszutauschen.

Otis – ENTstatistics ist eine an mehreren Schweizer Kliniken verwendete Software der Firma *innoForce Est.* Sie speichert die gesamte Krankengeschichte eines Patienten im Krankenhaus strukturiert ab und enthält damit sämtliche Daten, die der Aufbewahrungspflicht obliegen und für Abrechnungszwecke benötigt werden.

Diese Arbeit entwirft und implementiert ein neues Modul, das den Arzt bei der Arztbriefschreibung am Ende des Behandlungsprozesses unterstützt. Das Modul *Arztbriefgenerierung* in *ENTstatistics* erlaubt es dem Arzt als Anwender, per Mausklick aus den strukturierten Daten Text zu generieren und diese in einem Arztbrief darzustellen. Dazu kann er Vorlagen mit flexiblem Inhalt definieren und in diese je nach Empfänger andere Informationen einfließen lassen. Sowohl Vorlage als auch Brief können in einem WYSIWYG-Editor bearbeitet werden. Der Arztbrief wird intern als HL7-CDA-Dokument hinterlegt, einem standardisierten XML-Format für Arztbriefe. Außerdem kann er in die Formate PDF und RTF transformiert werden, falls andere Programme den CDA-Standard noch nicht unterstützen.

Abkürzungsverzeichnis

CDA	Clinical Document Architecture, HL7 Standard
DICOM	Digital Imaging and Communication in Medicine
DRG	Diagnosis Related Groups (deutsch Diagnosebezogene Fallgruppen)
ENT	Ear Nose Throat (deutsch HNO)
HNO	Hals Nasen Ohren (englisch ENT)
HL7	Health Level 7
ICD	International Classification of Diseases
IHE	Integrating the Healthcare Enterprise
KIS	Krankenhausinformationssystem
LGPL	GNU Lesser General Public License
MCS	In dieser Arbeit: Multicenterstudie; organisationsübergreifende Studie mit medizinischer Fragestellung.
OPS	Operationen- und Prozedurenschlüssel
OTST	*Otis Statistics*
WHO	World Health Organisation, (deutsch Weltgesundheitsorganisation)
WYSIWYG	What you see is what you get
XDT	Daten-Transfer-Standard (X für verschiedene Formate)
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation
XSL-FO	XSL Formatted Output

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1. Ausgangslage und Motivation

Die Behandlung eines Patienten ist nicht nur eine medizinische Angelegenheit für den behandelnden Arzt, sondern bringt durch mehrere Beteiligte auch einen erheblichen organisatorischen Aufwand mit sich.

Im Krankenhaus kann die Behandlung eines Patienten grob in die drei Schritte Aufnahme, Behandlung und Entlassung zusammengefasst werden, die je nach Blickwinkel um Vielfaches verfeinert werden können. Betrachtet man den administrativen Arbeitsaufwand des im Krankenhaus behandelnden Arztes ergeben sich die Teilschritte Dokumentation der Diagnosen, der Therapien und des Behandlungserfolgs und zuletzt das Verfassen des Arztbriefs. Der Arztbrief soll eine übersichtliche Zusammenfassung der durchgeführten Behandlung darstellen, die den Empfänger, also den weiterbehandelnden Kollegen, schnell, umfassend und verständlich informiert, um eine sachgerechte Kontinuität in der ärztlichen Betreuung zu gewährleisten (Semler, 2001).

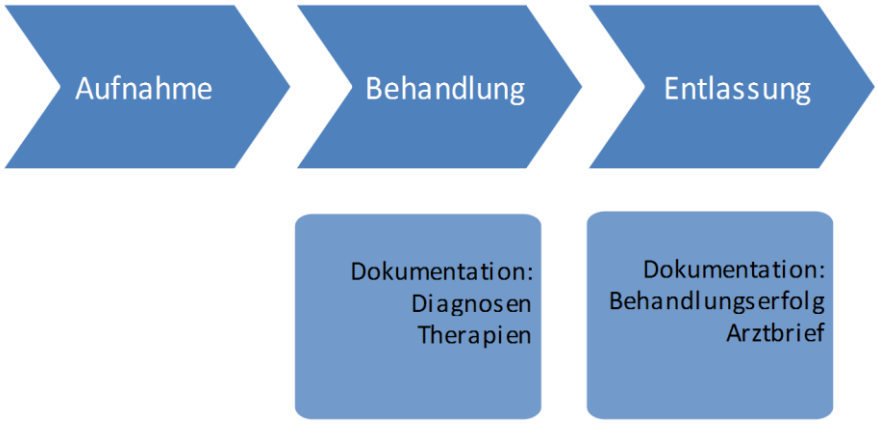


Abbildung 1: Geschäftsprozess - Krankenhausaufenthalt des Patienten[1]

Die Behandlung eines Patienten beginnt jedoch meist nicht erst mit der Aufnahme im Krankenhaus, sondern mit dem Gang des Patienten zum Arzt seines Vertrauens, meist der Hausarzt in einer niedergelassenen Praxis. Erst nachdem dieser feststellt, dass die Behandlung im Krankenhaus durchgeführt werden muss, erstellt er eine Überweisung und der Patient kann im Krankenhaus aufgenommen werden.

Die Mitwirkung mehrerer Personen verschiedener Organisationen an einem Fall (der Krankheit des Patienten) wird nur durch gut organisierte Abläufe und Informationsaustausch möglich. Abbildung 2 zeigt beim Hausarzt die Prozesse Untersuchung und Überweisung, woraus ein Überweisungsdokument entsteht. Beteiligte sind das Sekretariat und der Arzt selbst, die Daten werden je nach Praxis noch in papierbasierten oder elektronischen Akten abgelegt. Am Krankenhaus sind neben dem Arzt das Sekretariat bei der Aufnahme und Arzthelfer während der gesamten Behandlung beteiligt. Neben der elektronischen Akte gibt es meist zusätzlich die Papierakte, nach der Entlassung wird der Arztbrief zum Versand an den Hausarzt erstellt.

[1] Eigene Darstellung

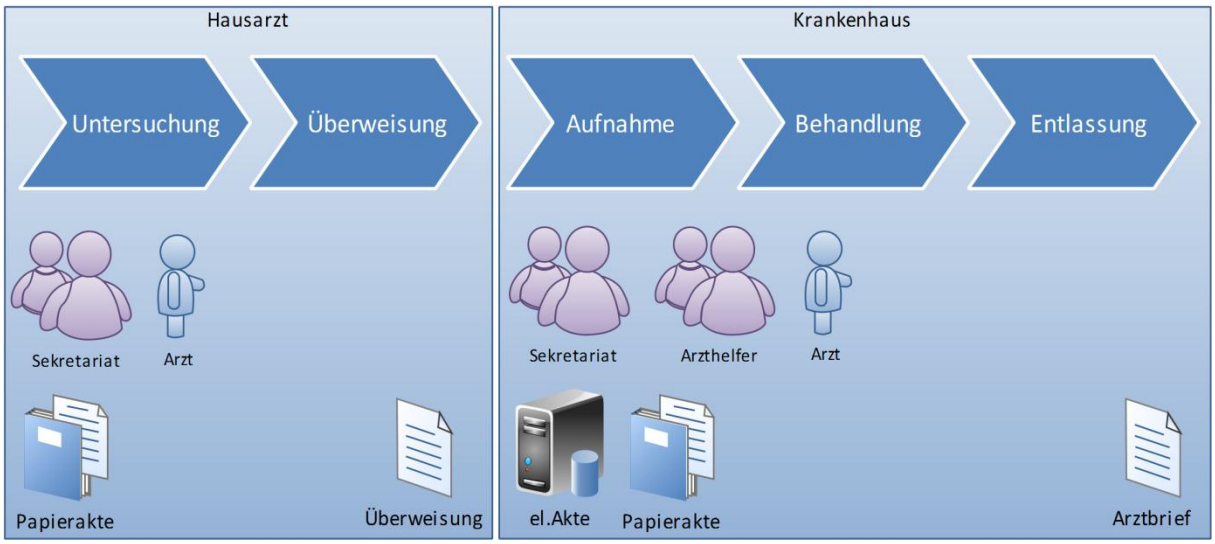


Abbildung 2: Geschäftsprozess Hausarzt, Klinik mit Beteiligen und Dokumenten[2]

1.2. Problemstellung

1.2.1. Probleme

Steigende Mobilität und dadurch vermehrter Arztwechsel, neue gesetzliche Anforderungen und neue Technologien ergeben hohe Anforderungen an die interne Organisation der Institutionen und die Kommunikation untereinander. Nachfolgend werden die Probleme aufgezeigt, die (mit Bezug zu Dokumentation und Arztbriefschreibung) bei den internen Aufgaben im Krankenhaus und der Kommunikation mit externen Stellen auftreten.

Problem 1

Krankenhäuser sind gesetzlich dazu verpflichtet, behandlungsrelevante Dokumente mindestens zehn Jahre aufzubewahren (Kantonsrat-Zürich, 2004). Eine reine papierbasierte Archivierung wird aus Platzgründen auf Dauer zu aufwändig und deshalb sind die Ärzte gezwungen, zumindest einen Teil der Daten elektronisch zu erfassen. Seit Beschluss der Einführung des diagnosebasierten Abrechnungsmodus in Krankenhäusern (in der Schweiz im Jahr 2009) sind diese außerdem gezwungen, für jeden Patienten detaillierte Daten zu gestellten Diagnosen und durchgeführten Therapien elektronisch zu erfassen, um mit den Kostenträgern abrechnen zu können.

Diese Arbeiten sind rein administrativer Natur und haben mit der medizinischen Behandlung der Patienten direkt nichts zu tun, die Erledigung jedoch verpflichtend, einmal durch den Gesetzgeber zwecks Dokumentation, einmal durch den Arbeitgeber zwecks Abrechnung. Der Arztbrief wiederum dient dem medizinischen Zweck, den Patienten nach Entlassung aus dem Krankenhaus optimal weiterbetreuen zu können. Er enthält üblicherweise die Informationen zu den gefundenen Diagnosen, durchgeführten Therapien und empfohlener Weiterbehandlung. Nicht ausgereifte Softwaretools zwingen den Anwender bei der Arztbriefschreibung zu Mehrfacherfassung der – zwar in anderer Form, aber vorhandenen – Daten. Dies führt einerseits zu unnötigem Zeitaufwand und andererseits zu geringerer Akzeptanz der Softwaretools bei Ärzten.

Problem 2

Bedingt durch den Zeitaufwand der Ärzte für verpflichtende administrative Arbeiten bleibt weniger Zeit für die eigentliche Arbeit: die medizinische Betreuung des Patienten und damit unter anderem für das Verfassen des Arztbriefs. Dadurch verzögert sich die Zeit von der Entlassung des Patienten bis zum Eintreffen des Arztbriefs beim Empfänger teilweise um bis zu 8 Wochen (Semler, 2001).

[2] Eigene Darstellung

Weitere Probleme

Der Arztbrief ist ein Dokument, mit dem sowohl der Arzt als auch das Krankenhaus nach außen auftreten – also aus Sicht des Krankenhauses gegenüber seinen Kunden (Patienten) und zuweisenden Ärzten, sozusagen den Lieferanten der Patienten. Einheitliches Layout, vollständige Informationen und zeitgerechtes Versenden können subjektiv zu einem besseren Image führen und damit auch einen wirtschaftlichen Einfluss auf das Krankenhaus haben. Für den Arzt ist der Arztbrief jedoch auch ein persönliches Dokument, in dem er auch seine eigene Note darin erkennen möchte, deshalb sollte nicht zu viel von oben herab vorgegeben werden.

Verschiedene Softwarehersteller verwenden verschiedene Technologien, Plattformen und Formate zur Speicherung ihrer Daten. An Kliniken ist es meist nicht möglich, alle Belange mit Produkten nur eines Herstellers zu bedienen und bei organisationsübergreifender Kommunikation (z.B. Klinik – Hausarzt) ist es unwahrscheinlich, dass beide dieselben Produkte verwenden. Ohne Standardisierung der Dokumentenformate und Softwareschnittstellen ist ein medienbruchfreier Datenaustausch nicht möglich.

1.2.2. Zieldefinition und Abgrenzung

Für die Therapiedatenbank der Firma innoForce soll ein Modul entwickelt werden, das den Aufwand des Arztes bei der Arztbriefschreibung verringert. Dabei soll auch der Wunsch des Krankenhauses zu einheitlichem Auftreten nach außen berücksichtigt werden. Um eine zukunftsfähige Lösung sowohl für das Krankenhaus als auch die Firma zu entwickeln, sollen die nationalen Bemühungen um Interoperabilität von Dokumenten im Gesundheitswesen berücksichtigt werden.

Ziele

Ziel 1: Verringerung des Aufwands bei der Arztbriefschreibung

Ziel 2: Klinikweit einheitliche Arztbriefe

Ziel 3: Standardisierte Dokumente (Interoperabilität)

Abgrenzung

Es soll ein Modul entwickelt werden, das die Arztbriefschreibung in einer bestehenden Software – der Therapiedatenbank der Firma innoForce – vereinfacht, also kein allgemeingültiger Weg zum automatischen Zusammenfassen von Daten gefunden werden.

Nationale Strategien zu mehr Interoperabilität im Gesundheitswesen sollen insofern berücksichtigt werden, sodass die erstellten Dokumente in standardisierter Form verfügbar sind. Die Übermittlung der Dokumente an andere Stellen ist nicht Thema dieser Arbeit.

1.3. Methodik und Aufbau der Arbeit

Zu Beginn (im Kapitel 2) behandelt die Arbeit Grundlagen zu den Themen Arztbriefe und Software in Krankenhäusern sowie Textgenerierung. Außerdem werden die technischen Details der bestehenden Software beschrieben, für die das Arztbriefmodul entwickelt werden soll sowie die verwendeten Technologien. Als Quellen dienen zum überwiegenden Teil Artikel aus Fachzeitschriften und Publikationen von staatlichen Stellen oder Organisationen im Informatik- und Gesundheitswesen. Originaldokumente zu anerkannten Standards werden meist von internationalen Gremien oder deren nationalen Ablegern herausgegeben und sind über das Internet erreichbar, teilweise aber nur für Mitglieder zugänglich.[3]

In den Kapiteln 3 und 4 zeigt die Arbeit nach der Entscheidung für Technologien und Standards den Prozess der Softwareentwicklung mit den Schritten Anforderungsanalyse, Entwurf der Use Cases und

[3] z.B. die vollständige HL7 Spezifikation

der grafischen Benutzeroberflächen, Aufteilung in Komponenten und Design der Klassen, schlussendlich die Implementierung.

Kapitel 5 fasst die Erkenntnisse aus dem Grundlagenkapitel zusammen, beschreibt die entworfene Systemarchitektur und beurteilt die implementierten Klassen. Am Ende werden im Ausblick die notwendigen Schritte skizziert, die vor Release des Moduls noch anstehen.

Der Anhang wird unterteilt in Teil 1 mit Screenshots der entwickelten Formulare und Teil 2 mit XML-Code. Der vollständige Quellcode ist aus rechtlichen Gründen nicht abgebildet, er gehört der Firma innoForce Est. Relevanter Code ist in den Kapiteln 3 und 4 ausschnittsweise abgebildet.

2. Grundlagen

2.1. innoForce

Dieses Kapitel gibt einen Überblick über die Firma innoForce und die funktionellen und technischen Grundlagen des Softwareprodukts, das im Rahmen dieser Diplomarbeit erweitert werden soll. InnoForce ist ein Ingenieurunternehmen, gegründet 2004 mit Sitz in Balzers im Fürstentum Liechtenstein. Die Firma entwickelt Software für Ärzte und Kliniken, ihre Kernkompetenzen liegen in den Bereichen Hals-Nasen-Ohren-Heilkunde (HNO), Hörgeräteakustik und Didaktik.

2.1.1. Produktportfolio (innoForce, 2011)

Otis – der virtuelle Patient

Das meistverkaufte Produkt – der virtuelle Patient – ist ein Lernprogramm für die Audiometrie-Ausbildung. Beim Audiometrieren wird die Hörleistung von Patienten festgestellt und aufgrund der Messungen Hörgeräte verschrieben und eingestellt. Der virtuelle Patient simuliert verschiedenste komplexe Hörschädigungen, sodass korrektes Audiometrieren selbstständig geübt werden kann, ohne einen Patienten belasten zu müssen. Dadurch verkürzt sich die Ausbildungsdauer und auch Fachpersonen erhalten die Möglichkeit, ihre Kenntnisse zu vertiefen. Die Kunden sind Ohrenärzte, Hörgeräteakustiker, Audiometrie-Schulen und HNO-Kliniken.

Otis – ENTstatistics

Die innoForce-Therapiedatenbank *ENTstatistics* wurde in Zusammenarbeit mit Otologen entwickelt (Otologie beschäftigt sich mit dem Ohr und seinen Erkrankungen). Mit diesem Datenbankprogramm lassen sich HNO-Therapien rationell erfassen und statistisch auswerten. Das Programm archiviert sämtliche Operations- und Folgeberichte, Audiogramme, Skizzen und Röntgenbilder. Die gespeicherten Therapiedaten lassen sich nach vielfältigen Kriterien abfragen und über prä- und postoperative Vergleiche sind Behandlungserfolge messbar. Zielgruppe der umfangreichsten Software der Firma sind HNO-Kliniken und niedergelassene Ohrenärzte, die Operationen durchführen.

Otis – Expertise Manager

Mit diesem speziell auf den Schweizer Markt zugeschnittenen Datenbankprogramm können Sozialversicherungsbeiträge bei Hörgeräteverordnungen berechnet werden. Die Software importiert die benötigten Ton- und Sprachaudiogramme automatisch von den Messgeräten, verhindert fehlende Angaben und erstellt die offiziell gültigen Formulare. Kunden sind Schweizer Ohrenärzte in niedergelassenen Arztpraxen und auch Kliniken, bei denen Patienten Hörschädigungen messen lassen.

Otis – AudioFit

AudioFit ist eine Software zum Abspielen von Geräuschen, mit denen der Hörgeräteakustiker beim Verkauf von Hörgeräten über 80 alltagsübliche Geräuschsituationen demonstrieren kann, um dem Kunden die Unterschiede zwischen verschiedenen Modellen aufzuzeigen bzw. um ihn zum Kauf eines Hörgeräts zu überzeugen.

Otis – AudiogramEdit

AudiogramEdit ist ein kleines Softwaretool zur einfachen Erstellung von Audiogrammen, es wird auch in den Programmen *ENTstatistics* und Expertise Manager verwendet.

2.1.2. innoForce-Therapiedatenbank (OTST[4])

Ein aktuelles Projekt der Firma ist eine Auftragsentwicklung für eine Schweizer Augenklinik, mit der sämtliche behandlungsrelevanten Daten elektronisch erfasst werden sollen. Die Software wird auf

[4] Otis Statistics

die bestehende innoForce-Therapiedatenbank aufgebaut (interne Bezeichnung OTST). Das in der Zieldefinition erwähnte Modul zur *Arztbriefgenerierung* ist ein Modul dieser Therapiedatenbank und Thema dieser Arbeit, dieses Kapitel beschreibt deshalb die für die Entwicklung des Arztbriefmoduls benötigten Grundlagen von OTST. OTST ist eine Software für HNO-Kliniken, die praktischen Beispiele kommen deshalb meist aus dem Fachbereich Ohrenheilkunde.

Technische Kennzahlen

Die Datenbank hinter OTST besteht aus über 130 Tabellen, die in Microsoft SQL Server (2000 oder später) verwaltet werden. Die Software wird in Delphi 2010 entwickelt und besteht aus zwei Anwendungen: dem Administrationstool und der eigentlichen Anwendung, dem Client. Das Administrationstool enthält rund 80 Klassen, der Client rund 90.

Krankengeschichte

Im Programm kann die gesamte Krankengeschichte und damit der Behandlungsablauf eines Patienten archiviert und übersichtlich dargestellt werden. Beispielsweise wird nach Aufnahme eines Patienten ins Krankenhaus eine Eingangsuntersuchung durchgeführt und erste Diagnosen gestellt. Nach weiteren Abklärungen in Form von Audiogrammen zur Messung der Hörleistung und Erstellung von Röntgenbildern wird eine Operation durchgeführt und deren Kenndaten im Operationsbericht eingetragen. Postoperativ werden wiederum ein Audiogramm und ein Bericht zur Nachuntersuchung erstellt. Durch die prä- und postoperativ durchgeführte Messung der Hörleistung kann der Behandlungserfolg einfach berechnet und im Programm visualisiert werden.

Abbildung 3 zeigt einen Screenshot einer beispielhaften Krankengeschichte.

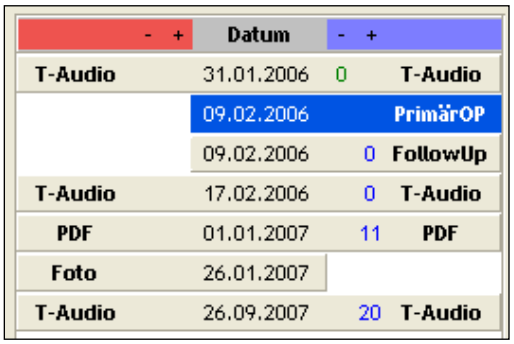


Abbildung 3: OTST - Krankengeschichte

Flexible Berichte

Die grundlegende Idee in OTST ist die Flexibilität der Berichte und das schnelle Erfassen dank einfacher Auswahl von vorkonfigurierten Kennwerten. Es besteht zwar ein Satz von fertig konfigurierten Berichten, aber jede Klinik kann im Administrationstool sowohl den möglichen Ablauf der Berichte als auch die Daten, die im Bericht erfasst werden können, anpassen. Der fertig konfigurierte Berichtsatz in OTST besteht für Ohrenkliniken aus einem Operations- und einem Nachuntersuchungsbericht. Im Operationsbericht sind die am häufigsten auftretenden Diagnosen und am häufigsten durchgeführten Therapien im Bereich Ohrenheilkunde eingetragen, im Nachuntersuchungsbericht Daten zum Allgemeinzustand des Ohrs, des Geschmackssinns und Gleichgewichtsgefühls – Kennzahlen aufgrund derer das Befinden des Patienten beurteilt werden kann.

Die Klinik definiert bei der Einführung der Software einmal die am häufigsten auftretenden Diagnosen und durchgeführten Therapien. Für die Dokumentation einer Operation muss der Arzt in der Software nun nur noch einzelne Felder aktivieren. Seltene Fälle können in Textfeldern dokumentiert werden bzw. kann die Berichtsstruktur im Administrationstool erweitert werden.

Die interne Struktur der Berichte kann man sich als Baum vorstellen, in dem flexibel Äste definiert werden können, die maximale Knotentiefe ist fünf. Abbildung 4 zeigt beispielhaft einen Teil der Diagnose- und Therapiezweige des Standard-Operationsberichts.

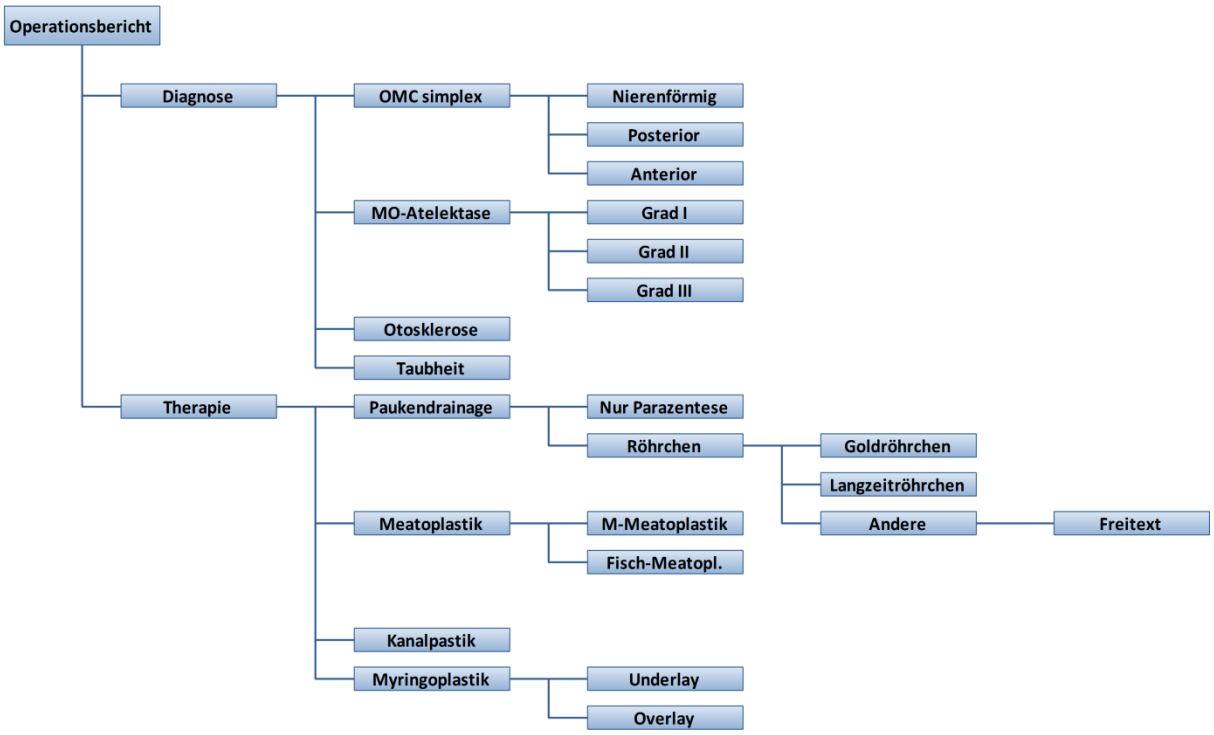


Abbildung 4: Baumstruktur als Datengrundlage für Berichte in OTST[5]

Bei der Erfassung des Operationsberichts aktiviert der Anwender die gewünschten Knoten der vorgegebenen Struktur. Abbildung 5 zeigt einen Ausschnitt des Fensters zur Berichtsdatenerfassung. Das Programm zeigt die Baumstruktur in mehreren Listenfeldern, von links nach rechts unten entsprechend den Ebenen im Baum.

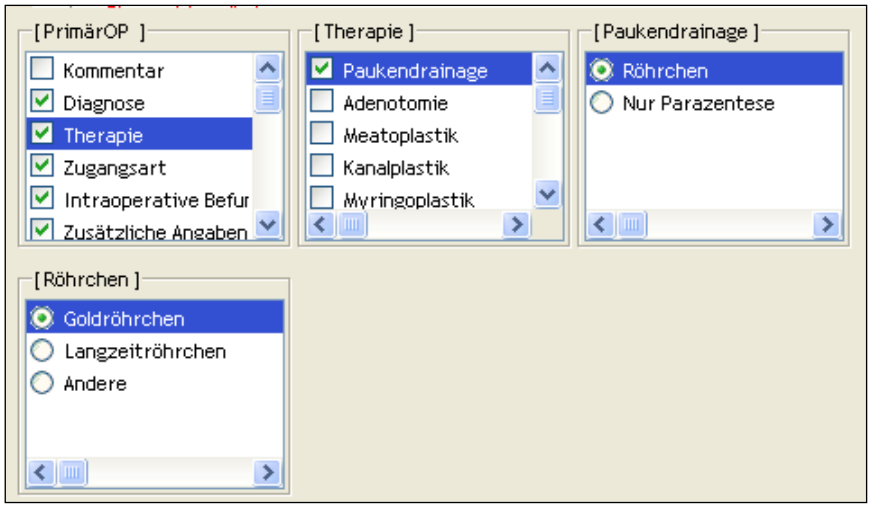


Abbildung 5: OTST – Erfassung von Berichtsdaten[6]

Im dargestellten Beispiel ist in der ersten Liste der Eintrag Therapie ausgewählt und als Folgeelemente Paukendrainage mit den Detaillierungen Röhrchen, Goldröhrchen. In der Baumstruktur in Abbildung 4 sind diese Einträge im zweiten Hauptzweig dargestellt.

[5] Eigene Darstellung
[6] Screenshot aus OTST

Abbildung 6 zeigt den zweiten Teil des Fensters zur Berichtsdatenerfassung und stellt im unteren Teil eine übersichtliche Zusammenfassung der erfassten Berichtsdaten dar. Als Therapieangaben sind das erwähnte Therapieelement Paukendrainage und ein weiteres aufgeführt.

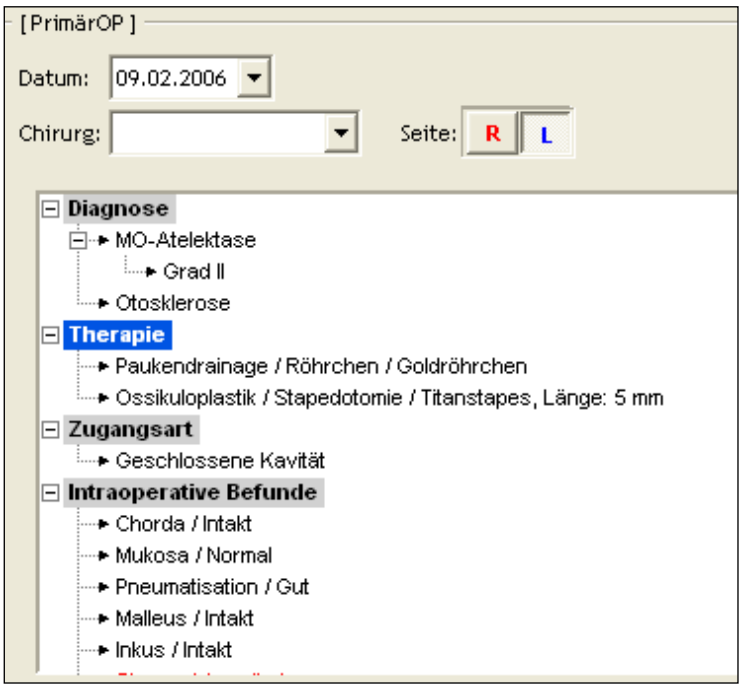


Abbildung 6: OTST – Berichtsdatenzusammenfassung[7]

Dadurch, dass die Berichte in Anzahl, sequentiellem Ablauf und Struktur verändert werden können, kann das Programm auf verschiedene Arbeitsabläufe angepasst werden. Es ist auch möglich, zu gespeicherten Bildern und PDF-Dokumenten Berichtsstrukturen zu erstellen und damit beispielsweise zu einem Röntgenbild eine Diagnose zu hinterlegen.

In der Datenbank werden die Konfiguration eines Berichts und seine Erfassung in jeweils sieben Tabellen gespeichert, Abbildung 7 zeigt die Beziehungen der Konfigurationstabellen mit Primär- und Sekundärschlüsseln, ohne die restlichen Datenfelder. In der Abbildung werden zur besseren Übersichtlichkeit die `Level4` und `Level5`-Tabellen nicht dargestellt, sie sind gleich aufgebaut wie die `Level1-3`-Tabellen und haben auch jeweils Fremdschlüsselbeziehungen auf die Vorgänger- und `Configuration`-Tabelle. In der Tabelle `TC_OPLC_Level_Configuration` wird eine Berichtsstruktur definiert, deren fünfstufige Baumstruktur wird in den Tabellen `TC_OPLx_Levelx` gespeichert. Eine Berichtsstruktur (z.B. Operationsbericht mit Diagnosen und Therapien) kann von mehreren Berichtstypen verwendet werden. So werden zum Beispiel für eine Primär-Operation und eine Folgeoperation (aufgrund Nicht-Erfolges beim ersten Versuch) dieselben Daten erfasst. Die Berichtstypen werden in der Tabelle `TC_OP_Operation` definiert, mit Verweis auf Berichtsstruktur-Tabelle.

[7] Screenshot aus OTST

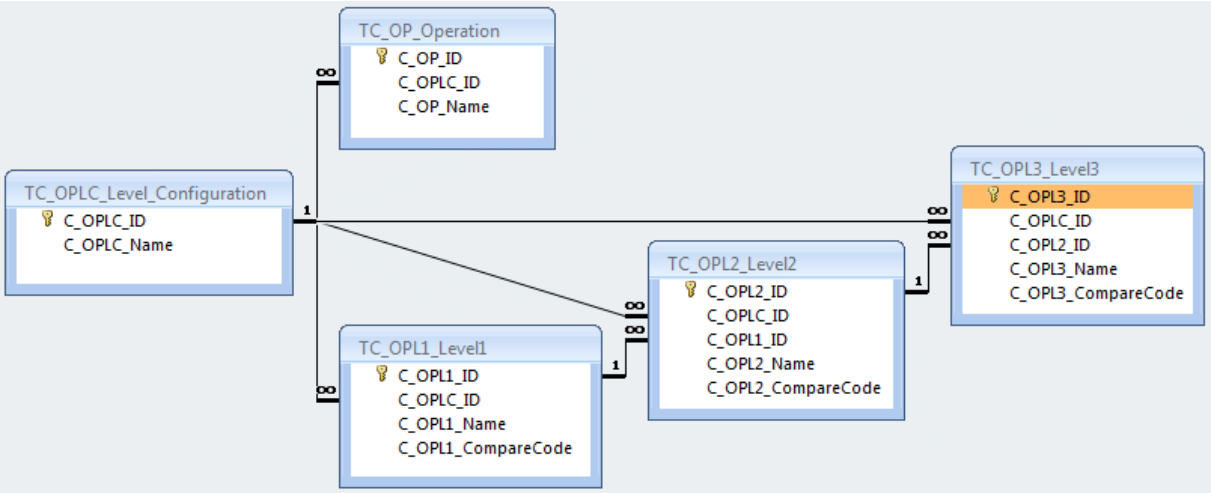


Abbildung 7: Tabellenstruktur (Teile) der Berichtskonfiguration[8]

Statistische Auswertungen

Durch Speicherung der gesamten Krankheitsgeschichte mit prä- und postoperativen Messwerten, Diagnosen und Therapiedaten können patientenübergreifend Behandlungserfolge berechnet werden. Das Statistikmodul der Software ermöglicht beliebige Filterkriterien, so kann für Ohrenkliniken beispielsweise die statistisch wahrscheinliche Verbesserung der Hörleistung bei Vorhandensein einer bestimmten Diagnose und Anwendung einer bestimmten Therapie berechnet werden.

Multicenterstudien

Multicenterstudien sind Studien mit medizinischer Fragestellung mit Daten aus verschiedenen Organisationen, also Kliniken. OTST erleichtert die Durchführung von Multicenterstudien, indem es den anonymisierten Export und Import von Daten unterstützt. Die teilnehmenden Kliniken müssen alle die Software verwenden, jedoch nicht zwingend dieselbe Konfiguration der Berichte aufweisen. Beispielsweise kann die erste Klinik die Diagnosen in einem Voruntersuchungsbericht erfassen während die zweite Klinik alles im Operationsbericht einträgt. Um den Datenaustausch zu ermöglichen, müssen sie gleichwertige Knoten ihrer Berichtstrukturen mit identischen Codes versehen, so genannten MCS-Codes. In den Tabellen sind sie hinterlegt als `C_OPLx_CompareCode`, siehe Abbildung 7.

2.2. Arztbriefe

Dieser Abschnitt behandelt das Thema Arztbriefe aus der Sicht des Mediziners und zeigt die Parallelen zu den administrativen Aufgaben, anschließend vorhandene Lösungen zur Unterstützung der Arztbriefschreibung von Konkurrenzsoftwareherstellern.

2.2.1. Allgemein

„Die Epikrise gehört unbestritten zu einer gut geführten Krankenakte als rückblickend geführte Darstellung des gesamten medizinischen Falles, die im nachstationären Weiterbehandlungsfall als Arztbrief, Entlassungsbrief oder Verlegungsbrief aufgesetzt und bezeichnet wird. Der Arztbrief (bzw. die Epikrise) soll somit eine kurze, überschaubare und übersichtliche Zusammenfassung der durchgeführten Patientenbehandlung und -betreuung darstellen, die zum einen dem Arzt dienen soll, Rechenschaft abzulegen über Verlauf und Behandlungsmanagement des Falles, zum anderen den Empfänger, den weiterbehandelnden Kollegen, in Stand setzen soll, sich schnell, umfassend und

[8] Eigene Darstellung

verständlich zu informieren und eine sachgerechte Kontinuität der ärztlichen Betreuung zu gewährleisten" (Semler, 2001).

Der Arzt am Krankenhaus erstellt den Arztbrief nach der Entlassung des Patienten, einmal intern als Zusammenfassung der Behandlung, einmal als externen Bericht, um den weiterbehandelnden Arztkollegen zu informieren.

2.2.2. Dokumentationspflichten

Archivierung

Krankenhäuser in der Schweiz sind gesetzlich dazu verpflichtet, behandlungsrelevante Daten mindestens zehn Jahre nach Behandlungsende aufzubewahren (Kantonsrat-Zürich, 2004). Die Art der Archivierung ist der Institution selbst überlassen. Es gibt grundsätzlich zwei Möglichkeiten bzw. drei bei Verwendung von beiden:

1. Papierbasiertes Archiv
2. Elektronisches Archiv

Die papierbasierte Archivierung ist bei Verwendung von papierbasierten Patientenakten nahe liegend, stellt die Krankenhausleitung bei hoher Patientenzahl jedoch mit der Zeit vor räumliche Platzprobleme und erschwert außerdem die Durchführung umfassender Statistiken.

Komplexe Bilddaten beispielsweise aus CT- (Computertomographie) oder MRT-Untersuchungen (Magnetresonanztherapie) bestehen aus einer Vielzahl von Einzelbildern und können nicht gesamthaft ausgedruckt werden bzw. verlieren beim Drucken auch an Informationsgehalt. Diese Daten können also sinnvoll nur elektronisch archiviert werden.

Das rein elektronische Archiv kann auf verschiedenen Wegen erreicht werden:

1. Einscannen der papierbasierten Patientenakte
2. Verwendung einer rein elektronischen Patientenakte

Das Einscannen der Papierdokumente ist letztendlich nur ein Ausweg aus dem Platzproblem, das durch die papierbasierten Archive auftritt. Der Vorteil von elektronisch vorhandenen Daten geht hier verloren, da grundsätzlich allein der Bezug Dokument-Patient vorhanden ist und es nur unter zusätzlichen Aufwand möglich ist zu erfassen, um welches Dokument es sich handelt bzw. welche Informationen im Dokument vorhanden sind.

Die vollständig elektronische Patientenakte ist in den meisten Kliniken noch Zukunftsvision, da der alltägliche Ablauf beispielsweise bei der Visite des Arztes am Krankenbett damit beginnt, dass die Arzthelfer die Dokumente für den Arzt heraussuchen bzw. zusammenstellen. Dieser verschafft sich vor dem Patientengespräch damit einen Überblick und macht während des Gesprächs Notizen zu den Aussagen oder weiteren Untersuchungen direkt in der Akte. Das ganze wäre theoretisch auch mit elektronischen Hilfsmitteln möglich, beispielsweise mit Hilfe von Tablet-PCs, jedoch sind die Klinikinformatik bzw. die Softwarehersteller noch nicht in der Lage, geeignete Softwaretools für alle Belange des klinischen Alltags bereitzustellen.

Abrechnung

Mit Einführung der DRG-Abrechnung (Diagnosis Related Groups) im stationären Bereich ab Januar 2012 (bzw. seit 2009 der Einführungsphase mit Datenerhebung an ausgewählten Kliniken) sind die Krankenhäuser dazu verpflichtet, detaillierte Diagnosen und Prozeduren (durchgeführte Therapien) zu jedem Fall elektronisch zu erfassen, um mit den Kostenträgern abrechnen zu können.

Das Prinzip der DRG-Abrechnung ist, dass jede Klinik für die Behandlung derselben Krankheit (identifiziert durch den genauen Diagnose-Code) dieselbe Vergütung erhält. Jeder Fall (Behandlung eines Patienten) wird in eine diagnosebasierte Gruppe eingeteilt, aufgrund derer die finanzielle Vergütung erfolgt. Die Berechnung der Gruppe erfolgt anhand der Hauptdiagnose, allfälligen

Nebendiagnosen und durchgeführten Prozeduren mit Hilfe eines DRG-Groupers, beispielsweise mit Hilfe des Webgroupers der SwissDRG AG (swissdrg, 2010).

Seit Einführung dieses Systems sind die Krankenhäuser quasi dazu gezwungen, Diagnosen und Therapien elektronisch zu erfassen um mit vertretbarem Aufwand in der Lage zu sein, die DRG-Codes zu berechnen und damit Geld von den Kostenträgern erhalten zu können.

2.2.3. Anwendersicht

Aus Sicht der Klinik ist ein einheitliches Layout der Arztbriefe wünschenswert, um nach außen – also gegenüber zuweisenden Ärzten, Patienten und Angehörigen – einheitlich aufzutreten. Übersichtliche Dokumente tragen zum positiven Auftreten einer Organisation bei und können durchaus einen Anteil zum Ruf des Krankenhauses beitragen; und da Patienten meist nicht an eine Klinik gebunden sind und deren Hausärzte frei zuweisen können, kann dies auch wirtschaftliche Folgen haben.

Aus Sicht des Arztes sollte der persönlich unterschriebene Brief auch eine persönliche Note haben und in Form und Layout nicht vollständig von oben diktiert sein. Es ist aber zeitraubend, wenn Informationen mehrfach erfasst werden müssen, beispielsweise Diagnosen und Therapien einmal für administrative Zwecke wie die Abrechnung und einmal für medizinische Zwecke wie Arztbriefe.

Wichtig für beide Seiten ist, dass alle benötigten Informationen vorhanden sind und keine zeitliche Verzögerung auftritt. Durch die weiter oben genannten gesetzlichen Pflichten gibt es im Bezug zum Arztbrief grundsätzlich zwei Lösungswege.

1. Manuelle Arztbriefschreibung und semantische Analyse mit Extraktion der benötigten Daten (Kaluschka, 2005).
2. Strukturierte Erfassung der benötigten Daten mit Textgenerierung.

Im Lösungsweg 1 verfasst der Arzt eine Zusammenfassung der Untersuchung in Textform, eine Software analysiert den Text und extrahiert daraus die strukturiert benötigten Kenndaten wie Diagnosen und Prozeduren. Im Lösungsweg 2 erfasst der Arzt die Daten in einer strukturierten Form und es besteht die Möglichkeit, daraus Text zu generieren und für Briefe zu verwenden.

Für den Arzt wäre die erste Lösung vom administrativen Aufwand die favorisierte, da hier nur ein Schritt zum Ergebnis – dem Arztbrief – benötigt wird. Hier ist aber schwer sicherzustellen, dass alle benötigten Informationen vorhanden sind bzw. ist dies erst nach der Extraktion der Daten technisch möglich. Aus dem Blickwinkel der Technik ist es nahe liegender, zuerst alle benötigten Informationen sicherzustellen und danach die Daten weiterzuverarbeiten. Auf diesem Wege ist es einfacher, Falscheingaben oder fehlende Daten direkt während der Erfassung zu verhindern.

2.2.4. Vorhandene Lösungen

Viele Anbieter von Software für Kliniken bieten eine Funktionalität zur Unterstützung bei der Briefschreibung. Im Funktionsumfang unterscheiden sie sich teilweise stark, es geht von reinen Textbearbeitungstools bis hin zum automatischen Einfügen von bereits erfassten Daten. Diese Aufzählung der Produkte ist bei weitem nicht vollständig zeigt jedoch, mit welchen Funktionalitäten die Hersteller für ihre Software werben und wie die Briefschreibungsfunktionalität in die Gesamtsoftware integriert ist.

AdOnco

AdOnco ist ein klinisch-wissenschaftliches Datenbanksystem zur Erfassung und Auswertung onkologischer Daten (Tumorerkrankungen) im Kopf-Hals-Bereich.

Die Brieffunktion wurde entwickelt, um die Akzeptanz der Software beim medizinischen Personal zu erhöhen. Es besteht die Möglichkeit, eine Zusammenfassung der onkologischen Anamnese (Vorgeschichte des Patienten mit Bezug zur Krankheit) vollautomatisch zu generieren und in Briefe einzufügen. Zusätzlich können Daten der überweisenden Ärzte verwaltet und eingefügt und Kopf- und Fußzeilen definiert werden. Die Briefe können direkt gedruckt oder in andere Textverarbeitungsprogramme exportiert werden. Weitere Funktionen können verschieden

detaillierte Darstellungen der onkologischen Daten generieren und in Briefe einfügen (Adunka, 2004).

iMedOne

Das Modul iMedOne.Client.MedText ist ein Modul für Arztbriefschreibung und Dokumentenmanagement des Krankenhausinformationssystems (KIS) der Firma tieto.

Die Funktionalität zur Arztbriefschreibung verfügt über Empfänger-Verwaltung, Serienbrieffunktion und Hilfsmittel zur Formatierung von Briefkopf und -fuß. Mit Hilfe von Platzhaltern können im Hauptsystem gekennzeichnete Informationen automatisch eingefügt werden, also neben dem Patientenstamm beispielsweise Diagnosen, Prozeduren oder auch verabreichte oder verschriebene Medikamente. Zusätzlich besteht die Möglichkeit, Drittsoftware zur Spracherkennung einzubinden und damit das elektronische Diktat zu ermöglichen. Erstellte Briefe können mit einem Status versehen werden, um den Arbeitsablauf im Krankenhaus abzubilden (tieto, 2011).

Phoenix

Phoenix ist das KIS der Firma MCS Parametrix Deutschland GmbH. Phoenix ermöglicht parametrierbare klinische Dokumentation für alle Fachabteilungen mit Einbindung von ICD-, OPS und DRG-Codes.

Die Funktionalität zur Arztbriefschreibung bindet MS Word ein und bietet automatischen Zugriff auf Patientendaten, Diagnosen, Therapien und Medikamente unter Verwendung von Word-Vorlagen. Einbindung von Systemen für digitales Diktieren und Spracherkennung, Signatur und Freigabe sowie Kommunikation mit Arztpraxen sind weitere beworbene Produktmerkmale (Computerführer, 2011).

2.3. Interoperabilität im Gesundheitswesen

Dieses Kapitel skizziert die Situation der verteilten Leistungserbringung im Gesundheitswesen und nationale Strategien für eine verbesserte Koordination und mehr Datenaustausch. Dazu werden auch die wichtigsten Standardisierungsorganisationen um im speziellen ein Standard für Arztbriefe beschrieben.

2.3.1. Allgemein

In der Schweiz gibt es mehr als 50 öffentliche Krankenhäuser (krankenhaus.ch, 2010) mit oftmals mehreren Kliniken, außerdem alleine in der Stadt St. Gallen (mit rund 72‘000 Einwohnern die siebtgrößte Stadt der Schweiz) mehr als 500 niedergelassene Arztpraxen (Gelbe-Seiten, 2010). Aufgrund der für Patienten meist freien Arzt- und Krankenhauswahl (je nach abgeschlossener Krankenversicherung) und der Mobilität der Bevölkerung wird ein Patient nur in den seltensten Fällen ausschließlich von einer Institution behandelt.

Für eine bestmögliche Beurteilung eines medizinischen Falles (also der Krankheit eines Patienten) ist es notwendig, dass der behandelnde Arzt über sämtliche vorhandene Informationen verfügt. Dies gestaltet sich, insbesondere bei einem Arztwechsel, aber auch bei einfachen Überweisungen von einer Praxis zur nächsten als schwierig, da verschiedene Institutionen ihre Daten in verschiedenen Systemen ablegen.

Um an die vollständigen Untersuchungsdaten einer anderen Institution zu kommen gibt es die folgenden Möglichkeiten:

1. Papierbasiertes Versenden per Post
2. Elektronischer Datenaustausch

Möglichkeit 1 gestaltet sich aus mehreren Gründen als kompliziert: Verzögerung im herkömmlichen Postweg, personeller und finanzieller Aufwand bei der Sender-Institution für das Zusammenstellen, Drucken und Versenden der Daten sowie Informationsverlust bei digital abgelegten Bilddaten.

Möglichkeit 2 ist theoretisch technisch möglich jedoch derzeit noch Zukunftsvision, die wegen mehrerer Probleme technisch noch nicht umsetzbar ist:

- Vielfalt der Softwaresysteme und dadurch
- Fehlende Interoperabilität von Dokumenten
- Fehlende Systeme zur Koordination des Datenaustausches

Weitere Probleme sind nicht technischer Art tragen jedoch auch dazu bei, dass Dokumente noch nicht problemlos zwischen Organisationen ausgetauscht werden können, beispielsweise die Frage, wer über die Rechte an Daten und Dokumenten verfügt und wie der Datenschutz geregelt ist.

2.3.2. Lösungsansätze

Strategie eHealth Schweiz

Der Schweizer Bundesrat hat 2007 mit der „Strategie ‚eHealth' Schweiz" das Ziel definiert, *„mit dem integrierten Einsatz von Informations- und Kommunikationstechnologien (IKT) zur Unterstützung und Vernetzung aller Prozesse und Teilnehmerinnen und Teilnehmer im Gesundheitswesen der Schweizer Bevölkerung den Zugang zu einem bezüglich Qualität, Effizienz und Sicherheit hoch stehenden und kostengünstigen Gesundheitswesen zu gewährleisten"* (BAG, 2007).

In der Schweiz ist das Bundesamt für Gesundheit (BAG) zuständig für eine Strategie für den Einsatz von Informations- und Kommunikationstechnologien im Gesundheitswesen und definiert als Ziel unter anderem*: „Bis Ende 2015 können alle Menschen in der Schweiz unabhängig von Ort und Zeit den Leistungserbringern ihrer Wahl den elektronischen Zugriff auf behandlungsrelevante Informationen ermöglichen (‚Elektronisches Patientendossier')"* (BAG, 2007, S. 4). Daraus ergeben sich Fragestellungen in mehreren Themenbereichen, unter anderem elektronische Gesundheitskarte, elektronische Zertifikate, digitale Signaturen, Datensicherheit oder Persönlichkeitsrechte der Patienten (eHealth, 2008, S. 2).

Die Planung und Umsetzung dieser Vision ist ein Thema, auf das in dieser Arbeit aus Platzgründen nicht weiter eingegangen werden kann. Die Komplexität wird in Abbildung 8 deutlich, sie zeigt die Basiskomponenten der von ehealthsuisse entworfenen Struktur für ein nationales Gesundheitsnetzwerk. Sie zeigt eine zentrale Infrastruktur zur Regelung von Sicherheit und Zugriffsberechtigungen. Patienten werden identifiziert über eine Versichertenkarte und den Patientenindex, Behandelnde über eine eigene Karte und einen weiteren Index. Dokumente werden dezentral gespeichert und es sind Zugangsportale sowohl für Behandelnde als auch für Patienten selbst vorgesehen. Auch ohne nähere Analyse der Komponenten wird klar, dass bei Interaktion mehrerer Komponenten die Schnittstellen klar definiert sein müssen und austauschbare Dokumente in standardisierter Form vorliegen müssen.

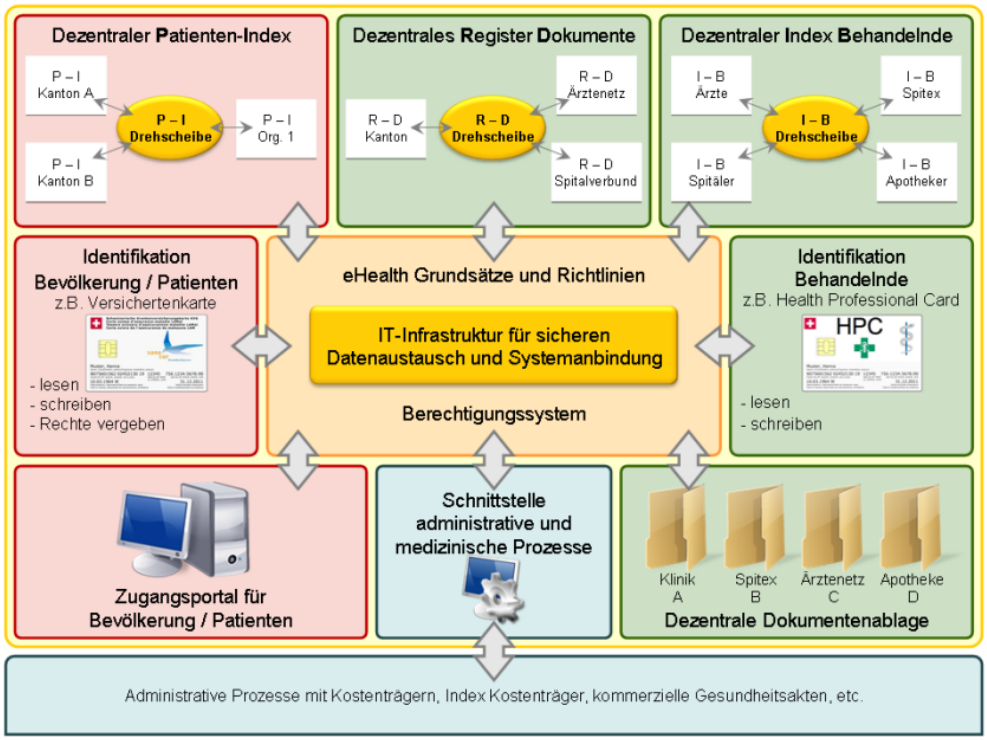


Abbildung 8: Basiskomponenten der Architektur „eHealth Schweiz“ (ehealthsuisse, 2009, S. 5)

Zukunftsvision

Nach flächendeckender Einführung einer elektronischen Patientenakte nicht nur in Kliniken sondern auch in niedergelassenen Arztpraxen ist ein Szenario denkbar, das vollkommen ohne Papier und Verzögerung im herkömmlichen Postweg auskommt und außerdem die vorhandenen Daten optimal nutzt.

1. Erfassung
   Alle Daten werden elektronisch erfasst und gespeichert. Die Organisation verwendet die Daten nicht nur für die Abrechnung (DRG) sondern auch für Statistiken für die strategische Planung, klinische Studien oder nationale Register.
2. Übermittlung
   Auf Daten, welche die weiterbehandelnde Organisation benötigt, kann sie nach Freigabe durch den Patienten zugreifen und daraus alle für sie interessanten Informationen entnehmen.
3. Speicherung
   Alle Daten werden in einer mindestens national standardisierten Form abgelegt bzw. es besteht die Möglichkeit, sie für den Export in Standardformate zu transformieren. Dadurch müssen einmal (in einer anderen Organisation) gemachte Untersuchungen nicht wiederholt werden, nur weil die Daten nicht oder nur in schlechter Qualität vorhanden sind.

IHE

„Die IHE (Integrating the Healthcare Enterprise) ist eine internationale Initiative zur Verbesserung des elektronischen Datenaustausches zwischen IT Systemen im Gesundheitswesen, indem sie die einheitliche Verwendung etablierter Standards wie HL7 oder DICOM vorantreibt“ (ihe-suisse, 2010).

Der Verein „IHE Suisse“ wurde im März 2010 als Schweizer Vertretung von „IHE Europe“ und „IHE International“ gegründet, um an den internationalen Bestrebungen partizipieren und auch davon profitieren zu können. Über die IHE Suisse wird auch sichergestellt, dass helvetische Besonderheiten als Erweiterungen implementiert werden und nicht zu Widersprüchen oder Blockaden führen (ihe-suisse, 2010, S. 1).

Die IHE Organisationen versuchen die Möglichkeiten des Datenaustausches zu erhöhen, zum Beispiel indem sie die Akteure (Kliniken, Softwarehersteller, Spezialisten) zusammenbringen, um Use Cases und Spezifikationen auszuarbeiten, nach deren Implementierung die verschiedenen Systeme besser interagieren (IHE, 2010). Aus Sicht eines Softwareherstellers ist es von Vorteil, Mitglied bei einer IHE Organisation zu sein, da er dadurch aktiv in die Prozesse eingebunden wird und nicht nur auf Vorgaben anderer Systeme reagieren muss.

2.3.3. Standardisierung

Es gibt mehrere Organisationen, die sich für die Standardisierung von Protokollen, Dateiformaten und Terminologien im Gesundheitswesen einsetzen. Nicht immer deren beste Unterstützer sind die Hersteller von Software, da die Standards meist recht aufwändig formuliert sind die Implementierung einen Zusatzaufwand darstellt, der auf den ersten Blick keinen Mehrwert bringt. Standards sind in der IT jedoch schon lange nicht mehr wegzudenken und eine Software, die allgemein anerkannte Schnittstellen und Dateiformate unterstützt ist für den Kunden attraktiver, da in größeren Organisationen mit einem Softwarehersteller alleine niemals alle Bedürfnisse abgedeckt werden können.

Standards im Gesundheitswesen können grob in drei Bereiche aufgeteilt werden, wobei nicht alle eindeutig einteilbar sind:

a. Dateiformate
b. Übermittlungsprotokolle
c. Terminologien

Beispiele:

Zu den Terminologien gehören die ICD- (herausgegeben von der Weltgesundheitsorganisation WHO) und OPS -Codes (herausgegeben vom DIMDI, dem deutschen Institut für medizinische Dokumentation und Information). ICD steht für „International Classification of Diseases“ und wird für die eindeutige Kodierung von Krankheiten verwendet (DIMDI, 2011), OPS steht für „Operationen- und Prozedurenschlüssel“ und wird für Kodierung von Therapien verwendet (DIMDI, 2011).

Dateiformate werden in mehreren Standards definiert, unter anderem in XDT, HL7 und DICOM. Haupteinsatzgebiet von DICOM („Digital Imaging and Communication in Medicine“) ist die Speicherung von Bilddaten (z.B. Röntgen). Im Standard wird neben dem reinen Dateiformat unter anderem auch das Kommunikationsprotokoll zum Datenaustausch beschrieben (NEMA, 2011). Der XDT-Standard wird im Gegensatz zu HL7 und DICOM hauptsächlich im niedergelassenen Bereich verwendet und steht für „X-Daten-Transfer“ und wird unterteilt in GDT (Geräte-), BDT (Behandlung-), ADT (Abrechnung-) und LDT (Labor-). Neben einem einfachen Dateiformat definiert auch dieser Standard die Kommunikationsbeziehung zwischen Sender- und Empfängersystem (KBV, 2011). HL7 steht für „Health Level Seven“ und wird im folgenden Abschnitt behandelt.

Health Level 7 (HL7)

Die wohl bekannteste Organisation, die sich mit Standards für die Interoperabilität von Software im Gesundheitswesen befasst, ist „Health Level Seven International“[9] mit über 55 nationalen Ablegern. Health Level Seven International ist eine von der ANSI (American National Standards Institute) akkreditierte „Standards Developing Organisation“ (SDO) im Bereich Gesundheitswesen, deren Domäne die klinischen und administrativen Daten sind (HL7, 2010).

Die neueste Version ist HL7 v3 und definiert im Gegensatz zu den noch oft verwendeten 2.x-Versionen Dateien im XML-Format. HL7 ist ein nachrichtenbasierter Standard, d.h. dass in den meisten Umgebungen das Sendersystem eine HL7-Nachricht (Datei) in einen definierten Ordner kopiert und das Empfängersystem sie einliest und löscht. Eine HL7-Datei wird immer definiert durch

[9] http://www.hl7.org

den Nachrichtentyp (z.B. Patientenaufnahme, Patientendaten ändern, Patient löschen), auf den das Empfängersystem verschieden reagieren muss.

Der Schweizer Ableger ist die„HL7 Benutzergruppe Schweiz“[10], ihre Mitglieder haben Zugriff auf den geschützten Bereich von HL7.org und damit Zugang zu allen Spezifikationen rund um HL7. Nach eigenen Angaben beteiligt sie sich aktiv an den nationalen Anstrengungen rund um die Strategie *„eHealth“ Schweiz. „Die Strategie ‚eHealth‘ Schweiz enthält mehrere Ziele, die aus unserer Sicht mit den Grundlagen von HL7 effizient, nachhaltig und länderübergreifend realisiert werden können.“* (HL7-Schweiz, 2010)

Im August 2010 hat HL7 Schweiz den Standard CDA-CH-II, die Spezifikation zum Erstellen von HL7-CDA Templates verabschiedet. Sie basiert auf der HL7 Clinical Document Architecture (CDA) Release 2 und wird im folgenden Abschnitt beschrieben (HL7-Schweiz, 2009).

CDA-CH-II

Die Schweizer Version der Clinical Document Architecture und wurde im August 2010 freigegeben. Sie enthält leichte Anpassungen gegenüber der Deutschen Version, die im Dokument VHitG-Arztbrief definiert wird (VHitG, 2006).

Kurz zusammengefasst ist ein CDA-Dokument ein Brief im XML-Format, standardisiert, damit der Empfänger die enthaltenen Informationen automatisiert einlesen und verwenden kann. Also er beispielsweise das Dokument erhält und der Brief automatisch einem Patienten zuordnet wird und medizinische Informationen wie Diagnosen und verordnete Medikamente in seine eigene Software übernommen werden. Die Darstellung des XML-Dokuments wird nicht im Standard definiert, eine Darstellung im HTML-Format zur Anzeigen im Browser ist jedoch mit Hilfe von XSL-Stylesheets einfach realisierbar.

„Ein CDA Dokument ist ein definiertes und komplettes Informationsobjekt, das Texte, Bilder und andere multimediale Objekte enthalten kann. CDA Dokumente beziehen sich auf einen Patienten und bilden administrative (Header) und medizinische (Body) Daten ab. Sie sind in der eXtensible Markup Language (XML) kodiert. Grundsätzlich gelten die Angaben aus der Spezifikation [CDA-CH]. CDA-CH macht nur Vorgaben zum CDA Header und lässt die Gestaltung des CDA Body weitgehend offen“ (HL7-Schweiz, 2009).

Ein CDA-Dokument muss gegen das XML Schema CDA.xsd[11] validieren. Der folgende Codeabschnitt zeigt das Wurzelelement mit Namensräumen und Grundstruktur (HL7-Schweiz, 2009, S. 12):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="vhitg-cda-v3.xsl"?>
<ClinicalDocument
        xmlns="urn:hl7-org:v3"
        xmlns:voc="urn:hl7-org:v3/voc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:hl7-org:v3 CDA.xsd">

  <!-- CDA Header -->

  <component>
    <structuredBody>

      <!-- CDA Body -->

    </structuredBody>
  </component>
```

[10] http://www.hl7.ch

[11] SupportingDocuments-v9\Schemas\CDA.xsd des CDA-CH-II Standards

Der CDA Header besteht aus fünf verpflichtenden Elementen zur Angabe von Dokumenttyp, Titel, Sprache, Datum und Vertraulichkeit. Weitere vier verpflichtende Elemente dienen der Angabe von Patientenstammdaten, Autor, Senderorganisation und Empfänger.

„Die eigentliche klinische Dokumentation wird im so genannten CDA Body festgehalten. Im Vordergrund steht hier ‚lesbarer' (narrativer) Text, der verpflichtender Bestandteil von CDA R2 Dokumenten ist und die Interoperabilität zwischen den menschlichen Kommunikationspartnern garantiert. Hier sind Möglichkeiten gegeben, diesen Text grob zu strukturieren, wie man dies von den Möglichkeiten der Textverarbeitung her kennt. Zur Strukturierung stellt die Standardspezifikation eine Reihe von XML-Elementen zur Verfügung, die als Body Structures zusammengefasst werden können. Der Body enthält ein oder mehrere Abschnitte (sections). Diese können auch ineinander geschachtelt sein, so wie Kapitel und Unterkapitel in einem Buch. Zudem sind Strukturierungen im Sinne von Tabellen oder Listen möglich" (VHitG, 2006, S. 24).

Der CDA Body kann unterschiedlich stark strukturiert sein, definiert durch die CDA Levels 1 bis 3. Level 1 konforme Dokumente zielen vor allem auf die menschliche Lesbarkeit ab und enthalten keine maschinenauswertbare medizinische Information. Level 2 konforme Dokumente sind in Abschnitte unterteilt, die von Applikationen ausgewertet werden können, die Abschnitte werden durch LOINC[12]-Codes gekennzeichnet. Beispielsweise könnten die Abschnitte eines Dokuments Anamnese, Diagnosen und Prozeduren sein. Level 3 konforme Dokumente enthalten maschinenlesbare Einzelinformationen, das heißt im vorherigen Beispiel wäre jede Diagnose oder Prozedur zusätzlich mit einem Code hinterlegt und damit automatisiert verwertbar.

2.4. Reporting[13]

Den Bereich Generierung von Text und Erstellen von druckfertigen Berichten deckt das Thema Reporting ab. Unter Reporting – auch betriebliches Berichtswesen – fallen die Maßnahmen und Hilfsmittel zur Erstellung und Speicherung von Informationen über einen Betrieb in Form von Berichten. Berichte sind die für eine genaue Zielsetzung zusammengefassten Informationen (Stichnote, 2006, S. 31).

Technisch gesehen ist Reporting das Abfragen und Zusammenfassen von Daten aus einer Datenquelle, dies kann eine Datenbank oder Dateistruktur sein. Gute Reportingprodukte sind in der Lage, aus verschiedenen Datenquellen zu lesen und ihre Berichte in verschiedene Formate zu exportieren und bieten außerdem flexible Berichtlayouts und -formatierungen, neben Text auch Grafiken und Tabellen.

Die zuvor analysierten Konkurrenzprodukte verwenden für die Arztbriefgenerierung keines der in diesem Kapitel behandelten Reporting-Produkte sondern setzen – soweit dies durch externe Analyse beurteilt werden kann – alle auf Eigenentwicklungen, die vollständig in die eigentliche Software eingebunden sind. Durch die Analyse von reinen Reporting-Produkten kann jedoch ein besseres Verständnis für die Möglichkeiten und den Umfang dieses Themas gewonnen werden.

2.4.1. Produkte für Reportgenerierung

Es gibt verschiedene kommerzielle und frei verfügbare Produkte für Reportgenerierung, von den bekanntesten werden in diesem Kapitel Funktionalität, Preis und Struktur beschrieben.

Crystal Reports (SAP)

Crystal Reports ist eine kommerzielle Software des Walldorfer Softwareunternehmens SAP zur Berichtsgenerierung aus Datenbanken oder Textdateien. Die Software ist Closed Source, das heißt, es kann nicht auf den Quellcode des Programms zugegriffen werden, außerdem bestehen Lizenzeinschränkungen im Bezug auf Verwendung von mit Crystal Reports erstellen Berichten oder

[12] Logical Observation Identifiers Names and Codes, http://loinc.org/
[13] Textgenerierung

Berichtstools in eigenen Programmen. Es besteht die Möglichkeit, Berichte in Java und .NET Applikationen einzubinden bzw. Standalone Anwendungen zu entwickeln, die Berichte aus bestehenden Datenbanken erstellen.

Crystal Reports ist in verschiedenen Editionen erhältlich (unter anderem Crystal Reports Server, Dashboard und Developer), die Preise liegen bei ca. 550€ für die Developer-Edition, die Server-Edition ist ab ca. 2'500€ erhältlich, je nach Anzahl Namens- oder Zugriffslizenzen (SAP, 2011).

JasperReports (JasperForge, 2010)

JasperReports ist eine Open Source Java Reporting Bibliothek mit Zugriffsmöglichkeiten auf verschiedene Datenquellen und Möglichkeit zur Generierung verschiedener Ausgabeformate (unter anderem PDF, RTF, HTML). Sie ist gratis verfügbar unter der LGPL-Lizenz, kann also in kommerzielle Software integriert werden. Das Prinzip bei der Generierung von Reports ist in Abbildung 9 dargestellt: Die Reports-Engine benötigt als Input die Berichtsdatei, die Datenquelle und die Parameterwerte für die Datenabfrage.

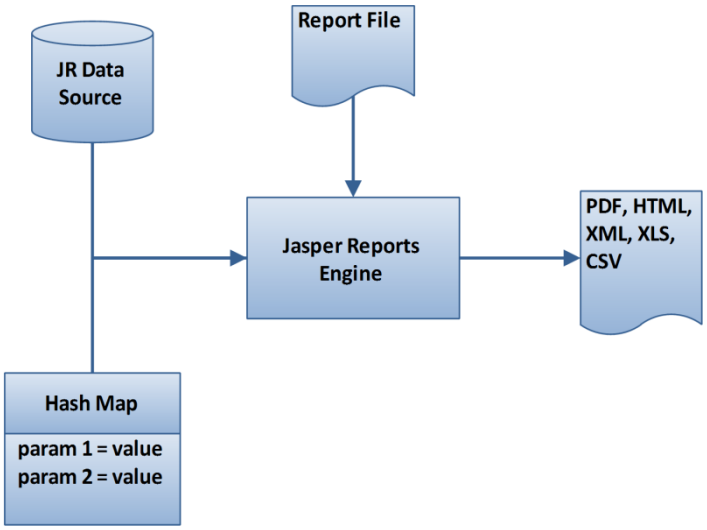


Abbildung 9: JasperReports Funktionsprinzip (JasperForge, 2010)

FastReports

FastReports Inc. bietet unter anderem eine kommerzielle Delphi-Komponente für Entwurf und Generierung von Reports an. Die Berichte werden in einem eigenen Tool erstellt und bearbeitet und können auch Formulare enthalten, mit denen beispielsweise vor Generierung zusätzliche Informationen vom Benutzer verlangt werden können. Als Datenquelle kommen alle mit SQL ansprechbaren Datenbanken in Frage.

2.4.2. Strukturen von Reportsprachen

Crystal Reports speichert seine Berichtsdateien verschlüsselt ab, sie können mit herkömmlichen Texteditoren nicht gelesen werden. Die anderen zwei analysierten Tools speichern ihre Berichtsdateien im XML-Format ab. Die Struktur der XML-Dateien ist ähnlich, so enthalten beide Elemente für Kopf- und Fußzeilen, die neben Textinhalt auch Formatangaben enthalten. Text wird in eigenen Elementen mit Formatangaben (Position, Schriftart- und Größe etc.) ähnlich platziert, die Referenzierung der Datenquelle unterscheidet sich jedoch. So kennt FastReports ein eigenes Element für SQL-Abfragen, in dem das SQL-Statement verschlüsselt hinterlegt wird und auf die die Textelemente verweisen, JasperReports fügt in die Textelemente `CDATA`-Ausdrücke mit der Bezeichnung der Datenfelder ein.

2.5. Technologien

Dieses Kapitel gibt einen Einblick in die verwendeten Technologien und Informationen zu weiterführender Literatur.

2.5.1. XML

XML ist das Dokumentenformat von HL7-CDA Dateien. Aus Platzgründen wird an dieser Stelle nur eine kurze Einführung in XML-Dokumente gegeben. Eine übersichtliche Einführung in das Thema XML gibt (Hauser, 2006).

XML steht für „eXtensible Markup Language“. XML ist eine Metasprache und ist aus der Informatik heute nicht mehr wegzudenken. *„Als Format von Dokumenten dient XML dazu, Daten zu speichern und zwischen unterschiedlichen Systemen austauschbar zu machen“* (Hauser, 2006, S. 11). XML wird vom W3C[14] definiert, die aktuelle Version ist 1.1 vom 16.08.2006 (W3C, 2009).

XML-Beispiel:

```
<?xml version="1.0" encoding="utf-8" ?>
<patients>
  <patient pid="9861">
    <surname>Mustermann</surname>
    <firstname>Max</firstname>
  </patient>
</patients>
```

Zu Beginn jedes XML-Dokuments steht immer die „Processing Insruction” zur Deklaration des XML-Standards. Ein XML-Dokument besteht aus Tags (Elemente) und Attributen. Elemente werden durch spitze Klammern „< >“ benannt und immer durch ein schließendes Tag beendet „</ >“. Es gibt immer nur ein Wurzel-Element. Ein Element kann beliebig viele Attribute enthalten. Attribute bestehen aus dem Namen und dem Wert, der immer in Anführungszeichen steht.

Datenmodell

Ein XML-Dokument kann als Baum mit einem Wurzelknoten und beliebigen Kind-Knoten gesehen werden. Programmiertechnisch lässt sich ein XML-Dokument deshalb leicht bearbeiten. Es gibt zwei Programmiermodelle:

- SAX Simple API for XML, De-facto-Standard zum Parsen von XML-Dokumenten
- DOM Document Object Model, W3C-Standard, Schnittstelle zum Zugriff auf XML

DOM stellt ein XML-Dokument vollständig als Baum mit Knoten dar und bietet Methoden zum Erstellen, Verändern, Löschen und Speichern des Objekts zur Verfügung. Das Prinzip von SAX ist das eventbasierte, serielle Durchlaufen eines XML-Dokuments. Dabei wird bei jedem Element oder Attribut ein Event ausgelöst, in dem programmiertechnisch Daten ausgelesen werden können.

Korrektes XML

Ein XML-Dokument gilt als wohlgeformt, wenn es allen XML-Regeln für Elemente und Attribute befolgt. Es gilt als valide (korrekt strukturiert), wenn es nur die in der Struktur erlaubten Tags und Attribute, ggfs. in der richtigen Reihenfolge verwendet. Die Struktur wird üblicherweise in einem DTD oder XML-Schema (XSD, „XML Schema Definition“) beschrieben. DTD steht für „Document Type Definition“ und ist von der Struktur einfacher als XML-Schema, jedoch nicht so mächtig und selbst keine XML-Datei.

Namensräume

Namensräume (engl. „Namespaces“, XMLNS, „XML Namespace“) dienen der Identifizierung von Elementen. Dies wird wichtig, sobald mit verschiedenen XML-Dokumenten gearbeitet wird, die

[14] World Wide Web Konsortium, http://www.w3.org/

eventuell gleichnamige Elemente mit unterschiedlicher Bedeutung enthalten. Ein Namensraum kann für ein Element und seine Unterelemente oder nur einzelne Elemente verwendet werden.

Namensraum-Beispiel:

```
...
<patients xmlns    ="http://www.mynamespace.com"
          xmlns:n1="http://www.yournamespace.com">
  <patient pid="9861">
    <surname>Mustermann</surname>
    <n1:phone>1234567</n1:phone>
...
```

Im Beispiel sind die Elemente `patient` und `surname` vom Namensraum des Attributs `xmlns`, das Element `phone` gehört zum Namensraum definiert im Attribut `xmlns:n1`.

2.5.2. XSLT

XSLT wird für die Transformation von XML-Dateien benötigt, im Modul *Arztbriefgenerierung* zur Transformation der HL7-CDA-Dateien in eine neue XML-Struktur. Dies wird sowohl von der Viewer-Logik als auch von der Export-Logik verwendet. Aus Platzgründen wird hier nur ein kurzer Einblick in die Idee von XSLT gegeben. Eine gute Einführung in das Thema XSLT gibt (Koch, 2007).

XSLT steht für „eXtensible Stylesheet Language Transformations“. *„Das Kernkonzept hinter XSLT ist die Transformation, mit XSLT können XML-Dokumente in andere XML-Dokumente oder XML-Formate wie HTML oder SVG umgewandelt werden“* (Koch, 2007, S. 125). XSLT wird vom W3C herausgegeben, die aktuelle Version ist 1.1 vom 5.12.2006 (W3C, 2006).

Ein XSLT-Dokument ist selbst ein XML-Dokument. Das Code-Beispiel transformiert das XML-Beispiel des vorangegangenen Kapitels in ein HTML-Dokument, wobei jeder Nachname in einer eigenen Zeile dargestellt wird.

XSLT-Beispiel:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http:/www.w3.org/1999/XSL/Transform"
  <xsl:output method="html" encoding="utf-8" indent="yes" />
  <xsl:template match="surname">
    <xsl:value-of select="." />
    <br/>
  </xsl:template>
</xsl:stylesheet>
```

Ein XSLT-Dokument wird immer durch das Wurzelelement `xsl:stylesheet` mit den angegebenen Namensraum definiert. Das `output` Element gibt hier an, dass ein HTML-Dokument ausgegeben wird, dadurch muss unter anderem das `html` Element nicht extra angeführt werden. Im `template` Element wird nach dem Elementnamen `surname` gesucht, und mit `value-of` der Wert ausgegeben, gefolgt vom HTML-Zeilenumbruch `br`. Die Referenzierung des Elementswerts mit dem Punkt im `select` Attribut des `value-of` Elements ist die einfachste XPath Referenzierung (mehr zu XPath weiter unten in diesem Kapitel).

Prozessor (Koch, 2007, S. 15)

Für die XSLT-Transformation wird ein Prozessor benötigt. XSLT-Prozessoren sind Software-Module, die an verschiedenen Stellen eingesetzt werden. Alle modernen Browser enthalten einen XSLT-Prozessor und die meisten höheren Programmiersprachen verfügen über Komponenten, um Transformationen durchzuführen. Nachfolgend beispielhaft zwei frei verfügbare Prozessoren:

- msxml DLL-Bibliothek von Microsoft, wird im Internet Explorer verwendet
- Xalan Opensource Prozessor der Apache Software Foundation (für Java / C++)

XPath

XPath (XML Path Language) ist die Sprache, mit der Elemente von XML-Dokumenten referenziert werden. In Zusammenhang mit XSLT hat XPath die Aufgaben Adressierung von Daten, Definition logischer Ausdrücke und Bereitstellung zusätzlicher Funktionen. XPath wird auch in anderen Sprachen, z.B. XQuery oder XPointer verwendet (Koch, 2007, S. 27).

XPath sieht alle Elemente (Attribute, Elemente, Namensräume) von XML-Dokumenten als Knoten an. Um einen Knoten zu referenzieren, wird vom aktuellen Knoten aus ein Pfad definiert. Es gibt sowohl absolute als auch relative Pfadangaben, der Pfad kann nicht nur baumabwärts (Kind-Knoten), sondern in jede beliebige Richtung weisen. XPath definiert die Richtungen als Achsen, so können z.B. Kind-, Eltern-, Vorfahr- oder Namensraumknoten referenziert werden.

Mit Hilfe der Operatoren können ausgelesene Werte je nach Datentyp kombiniert werden. Die wichtigsten Datentypen in XPath sind: Zahlen, Text, Knoten und Knotenmengen sowie Boolesche Werte (Koch, 2007, S. 42). Es gibt Arithmetische, Logische und Vergleichsoperatoren.

Über die XPath-Funktionen lässt sich die Transformation der XML-Ausgangsdaten steuern. Beispielsweise liefert die Funktion `name()` den Knotennamen und damit kann mit dem XSLT-Ausdruck `<xsl:value-of select="name(.)" />` der Wert des aktuellen Knotens ohne Kenntnis seines Namens ausgelesen werden. Es gibt Funktionen für alle Datentypen unter anderem für Zahlen eine Rundungsfunktion, für Strings Substring-Methoden.

Mehr Details zu XPath sind in (Koch, 2007) zu finden.

2.5.3. XSL-FO

XSL-FO wird von der Export-Logik dazu verwendet, die HL7-CDA-Dateien in Drittformate wie PDF zu transformieren. Eine Einführung in das Thema XSL-FO gibt (Krüger & Welsch, 2007).

Die „eXtensible Stylesheet Language – Formatting Objects“ beruht auf der W3C Recommendation XSL, deren Version 1.1 2006 publiziert wurde und ermöglicht zusammen den anderen Technologien rund um XML die Formatierung von Dokumenteninhalten in festen Seitendimensionen und damit druckfertige Aufbereitung von zuvor unformatiert gespeicherten Daten (Krüger & Welsch, 2007, S. 7).

Das XSL-FO-Konzept verwendet die folgenden Standards:

- XSLT Transformation der XML-Eingangsdaten
- XPath Adressierung der Strukturen
- XSL Formatierung in feste Seitendimensionen

Prozess

Die Transformation eines XML-Eingangsdokuments in ein Ausgabeformat ist ein mehrstufiger Prozess, Abbildung 10 zeigt den prinzipiellen Ablauf. Ein XML-Dokument wird mit Hilfe eines XSL-FO-Stylesheets durch den XSLT-Prozessor in ein XML-FO-Dokument transformiert. Der XSL-FO-Formatierer erstellt daraus und gegebenenfalls unter Verwendung zusätzlicher Dateien (Grafiken etc.) das Ausgabedokument (z.B. ein PDF).

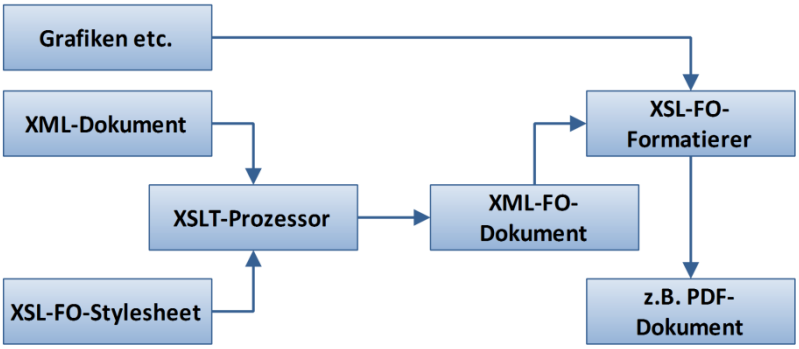


Abbildung 10: XSL-FO-Workflow (Krüger & Welsch, 2007, S. 16)

FO-Formatierer

Der XSL-FO-Standard spezifiziert eine Fülle von Konstrukten als XML-Elemente mit Attributen, mit denen Stylesheet und Layoutierung des Ausgangsdokuments festgelegt werden können. Die FO-Spezifikation beschränkt sich dabei auf Anweisungen, die vom FO-Formatierer umgesetzt werden müssen. Dies sind Softwareprodukte, von denen einige frei verfügbar, andere kommerziell sind. Mit dem Standard alleine können keine Dokumente erstellt werden, außerdem reicht er nicht aus, um praxistaugliche Ausgaben zu produzieren.

Beispiel:

```
..
<fo:block text-align="justify" hyphenate="true" language="de">
..
</fo:block>
..
```

In der Blockdefinition wird angegeben, den folgenden Text in Blocksatz mit deutscher Silbentrennung darzustellen. Der Standard liefert jedoch keine Hilfsmittel, wie Text als Blocksatz dargestellt oder Worte getrennt werden. Hier kommen die proprietären Funktionen der Formatierer ins Spiel, nachfolgend einige Produkte mit Herstellern:

- FOP Apache Software Foundation
- XEP RenderX Inc.
- XML2PDF Altsoft NV

FOP ist frei verfügbar und Opensource, Java-basiert und damit multiplattformfähig. Die aktuelle Version 1.0 ist in der Lage, neben PDF-Dokumenten auch RTF-Dokumente zu erstellen (Apache-Software-Foundation, 2010). Auch XEP basiert auf Java, ist jedoch ein kommerzielles Produkt, die Serverversion kostet rund 4‘000€ (RenderX, 2010). Der Formatierer von Altsoft basiert auf dem Microsoft .NET Framework und ist wie XEP als Server-, API und Desktopversion inkl. grafischer Oberfläche erhältlich. Die Serverversion ist hier ab rund 1000€ erhältlich (Altsoft, 2010).

2.5.4. Entwicklungsumgebung

Bei der Wahl der Entwicklungsumgebung ergeben sich bedingt durch die Firma und die bestehende Software keine Auswahlmöglichkeiten. Die bestehende Datenbank läuft auf Microsoft SQL Server (2000 oder später). Als Programmiersprache und Entwicklungsumgebung wird Embarcadero Delphi 2010 verwendet. Delphi ist eine objektorientierte Programmiersprache, die ursprünglich von Borland als Nachfolgersprache von Object Pascal entwickelt wurde.

Die Software wird ausschließlich für Microsoft Windows-Betriebssysteme ab Windows XP entwickelt.

2.5.5. Editoren

Das Softwaremodul *Arztbriefgenerierung* soll die Möglichkeit bieten, Arztbriefe in einem einfachen Editor zu bearbeiten. Die Speicherung der Briefe als XML legt die Verwendung eines HTML-Editors nahe. Dazu kann der XML-Brief per XSLT-Transformation in HTML und wieder zurück transformiert werden.

Für Editoren gibt es grundsätzlich zwei Lösungsansätze: WYSIWYG und WYSIWYM. Die Abkürzungen stehen für „What you see is what you get“ bzw. „… what you mean“ und umschreiben, wie der Anwender einen Text bei der Bearbeitung sieht. WYSIWYG-Editoren zeigen dem Benutzer formatierte Zeichen so an, wie sie beim Drucken (ungefähr) aussehen, WYSIWYM-Editoren stellen anders formatierte Zeichen gleich dar und zeigen nur, welchem Zweck sie dienen (z.B. fett, kursiv) (Preece, 1994, S. 723).

WYSIWYG-Editoren a la MS Word sind auch für technisch nicht versierte Personen leicht zu verstehen und eignen sich deshalb für medizinische Software, wo der Arzt als primärer Anwender gesehen wird, besser.

Die Standardkomponente von Embarcadero Delphi, um HTML-Seiten darzustellen ist `TWebBrowser`. Für das Delphi-Umfeld gibt es einige WYSIWYG-Editor-Komponenten, die allesamt auf die Einbindung dieser Komponente aufbauen. Sie stellen HTML-Code in der Internet Explorer-Komponente dar und bieten unterschiedliche Formatierungsmöglichkeiten an (fett, kursiv, Tabelle, Aufzählung etc.). Es folgt eine Beschreibung einiger analysierter Komponenten.

ProfDHTMLEdit

`ProfDHTMLEdit` ist eine kommerziell erhältliche Delphi-Komponente von ProfGrid.Com (http://www.profgrid.com/dhtmledit.html). Sie kostet mit Sourcecode als Einzelentwicklerversion rund 200€ (ProfGrid, 2010). *„The ProfDHTMLEdit component is just about the easiest way to give your applications the familiar capabilities of WYSIWYG editing, while taking advantage of the DHTML-rendering capabilities available in Microsoft Internet Explorer 5.0 and higher“* (ProfGrid, 2010).

Die Komponente ist als Demoversion ohne Sourcecode inklusive Beispiele gratis erhältlich. Sie bietet alle grundlegenden Formatierungsmöglichkeiten, die man aus kleineren Texteditoren kennt:

- Schriftart, Schriftgröße
- Text Fett, Kursiv, Unterstrichen
- Tabellen, Listen (nummeriert, nicht nummeriert)

Als Eingabeformat wird ein HTML-Dokument verlangt.

EmbeddedWB

EmbeddedWB ist eine freie Delphi-Komponente von Bsalsa Productions. *„Bsalsa productions develop freeware solutions for users and programmers with ‚Embarcadero Delphi' for 'Microsoft Windows' OS. Our main product is the Embedded Web Browser Component Pack“* (Bsalsa, 2009). Das Package Embedded Web Browser enthält die WYSIWYG-Editor-Komponente `EmbeddedWB` und wurde bis Delphi 2009 entwickelt, es sind also gegebenenfalls Anpassungen für Delphi 2010 notwendig.

Die Komponente ist frei und ohne Lizenzeinschränkungen inklusive Sourcecode erhältlich. Sie bietet ähnliche Textbearbeitungsfeatures wie `ProfDHTMLEdit`.

HTMLEditor

Die HTMLEditor-Klassen von theunknownones.googlecode.com sind WYSIWYG-Editoren für Delphi 2010 (theunknownones, 2010). Sie sind frei erhältlich unter der Lizenz LGPL (GNU Lesser General Public License). Die Lizenz erlaubt die Verwendung der Komponente bei Beibehaltung der eigenen, auch proprietären Lizenz (FSF, 2007).

Die Klassen sind in Versionen für Delphi 2009 und 2010 vorhanden und bieten ähnliche Möglichkeiten wie die anderen genannten.

Beurteilung

Ob eine Komponente sich für die Einbindung in die Software OTST eignet, hängt von mehreren Faktoren ab:

- Lizenz
- Funktionalität und Support
- Weiterentwicklung
- Preis

Die Lizenz muss erlauben, die Komponente in die proprietäre Software OTST einzubinden. Die Funktionalität muss den Anforderungen genügen. Bei Problemen während der Implementierung ist

es von Vorteil, Support von einer Firma oder über ein frequentiert besuchtes Forum zu bekommen. Die Komponente sollte für die aktuelle Delphi Version vorhanden sein und – sofern beurteilbar – auch in naher Zukunft noch vom Hersteller betreut werden. Der Preis sollte unter 300€ liegen (Vorgabe der Firma).

3. Anforderungen und Entwurf

Die Anforderungsanalyse dient dazu, die Bedürfnisse der verschiedenen Interessensgruppen (Kunde, Anwender, Hersteller) zu dokumentieren. Daraus ergeben sich die funktionalen und technischen Anforderungen, woraus die Use Cases definiert werden können. Am Ende steht eine interne modulare Aufteilung des neuen Submoduls *Arztbriefgenerierung* mit Schnittstellen zur bestehenden Software OTST.

3.1. Anforderungen

Die Anforderungen an das Softwaremodul sind von allen Seiten nur grob definiert, die detaillierten Anforderungen werden im Rahmen der Planung und Entwurf des Moduls in den folgenden Unterkapiteln beschrieben.

Anwender

Anwender der Software sind Ärzte oder ärztliches Personal und nicht ausschließlich Informatiker. Die Software muss also ohne erweiterte technische Kenntnisse bedienbar sein. Das Modul *Arztbriefgenerierung* muss die Möglichkeit bieten, Vorlagen für Briefe zu definieren, in die verschiedene Daten einfließen können. Beim Erstellen des Briefes aus einer Vorlage soll der Brief nachbearbeitet und in externe Programme exportiert werden können.

Kunde

Zahlender Kunde ist die Krankenhausleitung. Ihr Interesse liegt im einheitlichen Auftreten in Dokumenten und vermindertem zeitlichem Aufwand für das Personal durch das Verwenden der Software.

Hersteller

Das Modul *Arztbriefgenerierung* muss in die bestehende Software OTST integriert werden und in der Lage sein, eine Zusammenfassung der Krankheitsgeschichte eines Patienten zu generieren, die als Arztbrief verwendet werden kann. Zum Definieren des Briefinhalts soll eine Art Grammatik definiert werden, mit der es möglich ist, Datenmengen aus der bestehenden Programmstruktur zusammenzufassen und formatiert auszugeben. Neben Stammdaten (Patienten-, Arzt- und Versicherungsdaten) sollen insbesondere die strukturiert abgelegten Berichtsdaten einfließen können. Dabei ist zu beachten, dass deren Struktur flexibel ist.

Das Briefgenerierungsmodul könnte sowohl für bestehende Anwender von OTST als auch für potentiell neue Kunden der Firma interessant sein, es soll deshalb darauf geachtet werden, dass die Implementierung zukunftsfähig gemacht wird. Einige aktuelle Projekte von HL7 Schweiz zielen auf die Interoperabilität von Dokumenten mit Verwendung von HL7-Standards ab. Um auch künftig den Kunden alle technischen Möglichkeiten bieten zu können, sollen dieselben Standards verwendet werden.

3.1.1. Entwurf der Use Cases

Ein Use Case (englisch für Anwendungsfall) beschreibt den Ablauf, den ein Anwender in einem System durchlaufen muss, um ein bestimmtes Ziel zu erreichen. Für das Modul *Arztbriefgenerierung* ergeben sich sechs Use Cases, wobei sich jeweils die Hälfte im Administrationsprogramm (Admin-Tool) und der eigentlichen Anwendung (Client-Tool) abspielen (siehe Abbildung 11).

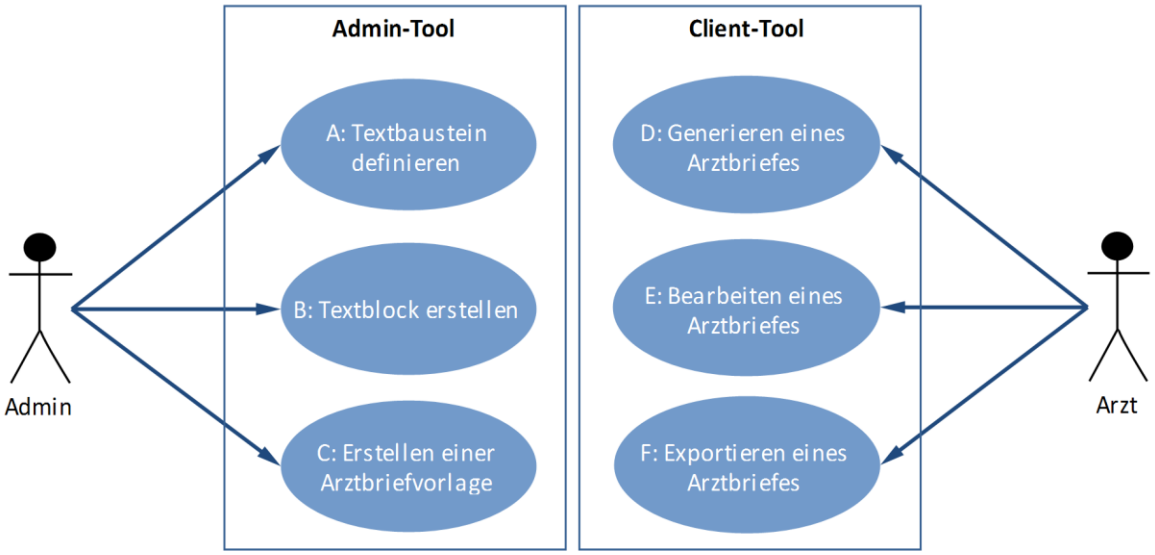


Abbildung 11: Übersicht der Use Cases[15]

Im Admin-Tool werden Textbausteine zu Berichtselementen definiert (A), damit aus diesen bei der Generierung des Arztbriefes (D) Fließtext generiert werden kann. Eine Arztbriefvorlage (C) definiert Form und Inhalt des Briefes und kann aus zuvor definierten Textblöcken (B) zusammengesetzt werden. Nach Generierung des Briefes aus einer Vorlage (D) wird der Brief durch den Anwender im Client-Tool bearbeitet (E) und in ein anderes Format exportiert (F).

Ein detailliertes Verständnis der Use Cases ist an dieser Stelle noch nicht möglich. Vor ihrer detaillierten Ausarbeitung (in Kapitel 3.4) werden an dieser Stelle im ersten Teil des Entwurfs die notwendigen technischen Details erläutert.

3.2. Basisarchitektur

Damit der Leser in der Lage ist, diese Arbeit sequentiell durchzulesen und für das Verständnis eines Kapitels nicht ein späteres vorausgesetzt wird, wird anstelle der Ausarbeitung der Use Cases an dieser Stelle der erste Teil des Entwurfs eingeschoben.

Grundlegend für das Verstehen des Softwaremoduls sind das Prinzip der Vorlagen und die Idee der Textgenerierung.

3.2.1. Vorlagendefinition

Eine Briefvorlage definiert neben dem Briefinhalt das Format des Briefes mit Kopfzeile, Fußzeile und Schriftart. Der Briefinhalt wird unterteilt in allgemeine Daten und Brieftext. Unter die allgemeinen Daten fallen alle im CDA Header verpflichtend anzugebenden Elemente, die nicht erst bei der Generierung bekannt werden (z.B. Dokumententyp). Der Brieftext enthält die eigentliche Information des Briefes mit Freitext und Platzhaltern.

Um die Definition von Briefvorlagen für den Benutzer zu vereinfachen, sollen Textblöcke definiert werden können, aus denen der Brieftext zusammengesetzt werden kann. Dies bringt folgende Vorteile:

1. Einmalige Definition von Blöcken, mehrfache Verwendung in Briefvorlagen.
2. Ausblenden der technischen Details bei der Vorlagendefinition durch einfache Auswahl von Textblöcken.
3. Einfachere technische Unterstützung des CDA Body Levels 2 durch Zuweisung eines Section-Identifiers (siehe Kapitel 2.3.3) zum Textblock.

[15] Eigene Darstellung

Abbildung 12 skizziert die entworfene Struktur der Vorlagen mit den Verweisen auf Textblöcke.

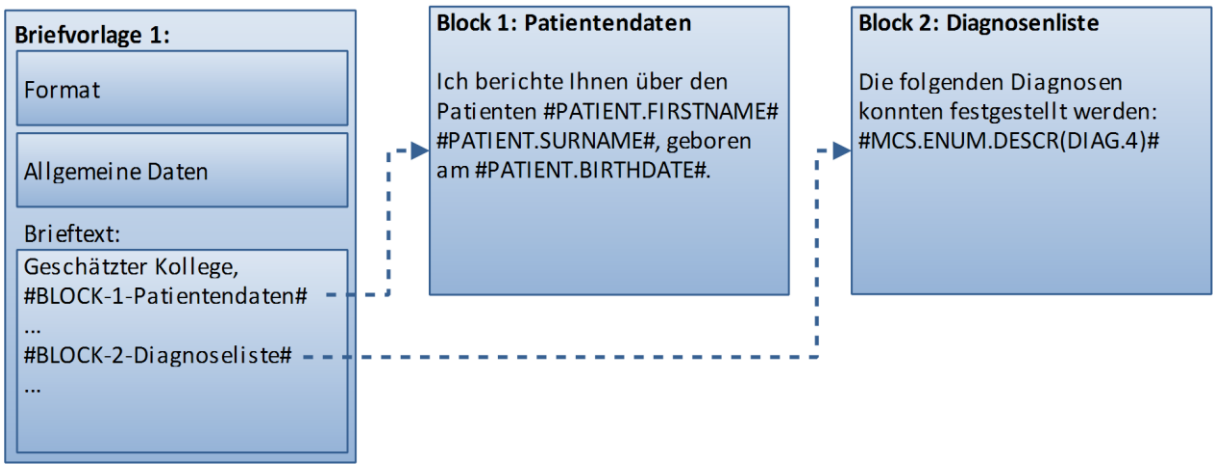


Abbildung 12: Briefvorlage mit Textblöcken[16]

3.2.2. Textgenerierung

Grundsätzliche Idee der Textgenerierung für OTST ist die Definition von Platzhaltern, die in der Briefvorlage eingesetzt und bei Generierung des Briefes durch reale Daten ersetzt werden (siehe Abbildung 13).

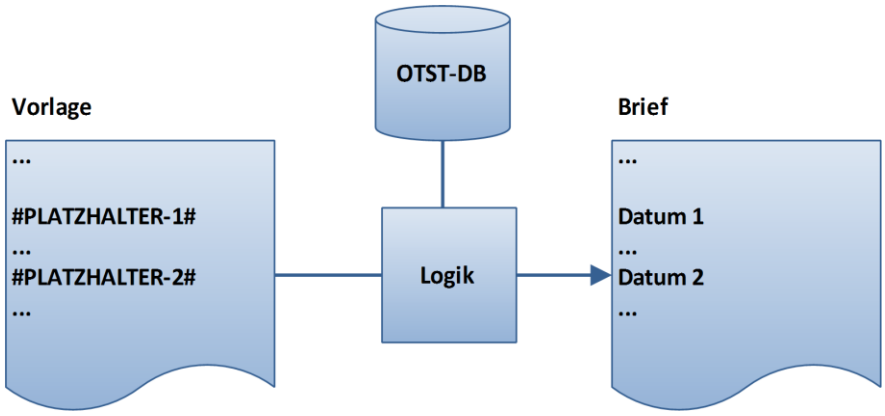


Abbildung 13: Ersetzen der Platzhalter[17]

Durch die Struktur der OTST-Datenbank und die Anforderungen ergeben sich für die Platzhalter zwei Kategorien:

1. Stammdaten
2. Berichtsdaten

Stammdaten sind fixe Daten von Patienten oder anderen Personen (z.B. Nachname oder Straße). Berichtsdaten sind Daten aus den im Kapitel 2.1.2 innoForce-Therapiedatenbank (OTST) beschriebenen flexiblen Berichtstrukturen.

[16] Eigene Darstellung
[17] Eigene Darstellung

Platzhalter für Stammdaten

Für die Stammdaten werden Platzhalter für die folgenden Personen zur Verfügung gestellt: Patientendaten und Daten des Brief erstellenden Arztes. Die Realisierung mit Platzhaltern gestaltet sich voraussichtlich relativ einfach, zu jedem Ausdruck kann eine SQL-Abfrage hinterlegt werden.

Tabelle 1: Generierbare Stammdaten

Datenquelle	Daten
Patient	Nachname, Vorname, Geburtsdatum, Straße, Stadt, Postleitzahl
Arzt	Titel, Nachname, Vorname

Platzhalter für Berichtsdaten

Es soll möglich sein, aus den aktivierten Elementen eines Berichts wahlweise eine Aufzählung in Listenform oder Textform zu generieren. Da die Knoten nur mit einem einfachen Bezeichner beschrieben werden, muss für die Generierung von Text zusätzlich zu jedem Knoten ein Textbaustein hinterlegt werden. Das Zusammensetzen der Textbausteine in Fließtext soll von der Software automatisch übernommen werden, ohne dass der Benutzer darauf Einfluss nehmen kann bzw. muss.

Grundsätzliche Idee ist die Wiederverwendung der MCS-Codes (jeder Knoten einer Berichtsstruktur verfügt über einen eindeutigen MCS-Code, siehe Kapitel 2.1.2 - Multicenterstudien). Ein Berichtsteil – also der Ast eines Baums oder ein einzelnes Knotenelement – wird im Platzhalter mit Hilfe des MCS-Codes referenziert. Hinzu kommen Angaben zu Inhalt, Formatierung und ob eventuell vorhandene Kind-Elemente verarbeitet werden sollen. Die Logik generiert nur Text für Platzhalter, welche im Bericht aktivierte Knoten referenzieren. Nicht aktivierte Elemente ergeben folglich Leertext.

Ein Platzhalter für ein Berichtselement besteht aus den vier Elementen Formatierung, Inhalt, MCS-Code und Anzahl Kind-Ebenen. Tabelle 2 beschreibt die möglichen Werte.

Tabelle 2: Elemente des Platzhalters für Berichtsdaten

Element	Möglichkeiten
Formatierung	Aufzählung als Liste, Aufzählung als Fließtext
Inhalt	Name des Elements, Textbaustein des Elements, Wert des Elements (Zahlwert, Datum)
MCS-Code	Eindeutiger Code innerhalb der OTST-Datenbank
Anzahl Kind-Ebenen	0 bis 4

Aus der Idee der Textgenerierung mit Platzhaltern ergibt sich an späterer Stelle die Grammatik (siehe Kapitel 3.5.1)

3.3. Komponentenstruktur

Nach Ausarbeitung der Anforderungen und erstem Entwurf zeichnet sich die angestrebte Architektur der Software ab. Die Systemarchitektur teilt das Softwaremodul in Komponenten auf.

Das Ziel dieser Aufteilung ist, einzelne Programmteile zu entwerfen, die klar definierte Aufgaben übernehmen und deren Interaktion klar geregelt ist. Dies bringt für die Implementierung Vorteile in mehreren Bereichen:

Klar definierte Komponenten vereinfachen die zeitliche Abschätzung für den Implementierungsaufwand bzw. machen sie nachvollziehbar und dadurch realistisch. Das Testen bzw. die Fehlersuche wird vereinfacht, da nicht nur die Gesamtfunktionalität, sondern durch Tests der Komponenten Teilfunktionen einfacher überprüft und damit Fehler leichter gefunden werden können. Werden die Schnittstellen frühzeitig definiert, könnten die Komponenten von verschiedenen Personen implementiert werden (Arbeitsaufteilung). Nicht zuletzt kann eine Komponente zu einem späteren Zeitpunkt leicht ausgetauscht werden, ohne Einfluss auf die bestehende Gesamtfunktionalität zu nehmen.

Funktionale Aufteilung

Von der Funktion gesehen kann das Softwaremodul in drei Komponenten aufgeteilt werden:

1. Generator
2. Viewer
3. Transformator

Der Generator generiert aus der Vorlage den Brief, im Viewer wird sowohl der Brief als auch die Vorlage vom Anwender bearbeitet. Der Transformator schlussendlich kümmert sich um die Transformation zwischen den verschiedenen Formaten. Abbildung 14 zeigt die Komponenten mit Eingabe und Ausgabe.

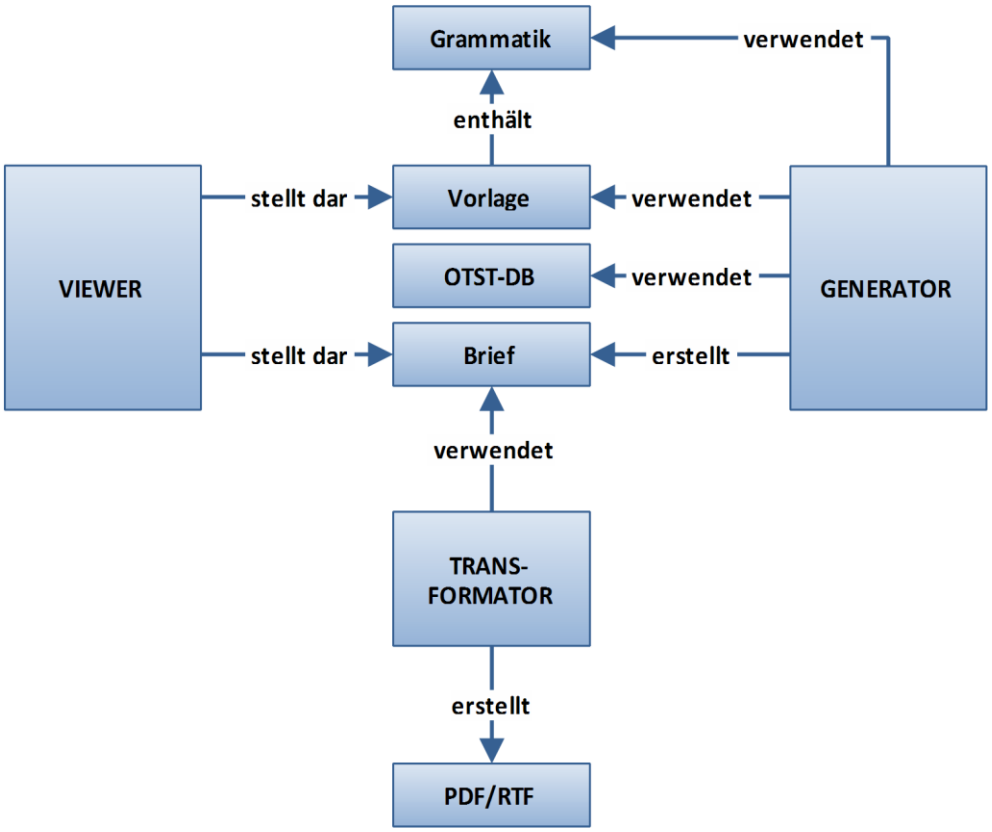


Abbildung 14: Funktionale Aufteilung in drei Komponenten[18]

[18] Eigene Darstellung

3.4. Use Cases

„Ein Use Case (englisch für Anwendungsfall) beschreibt anhand eines zusammenhängenden Arbeitsablaufs die Interaktion mit einem technischen System. (…) Er beschreibt Anforderungen an das System, d.h. was es leisten muss, aber nicht wie es dies leisten soll. (…) Ein Anwendungsfall wird stets durch einen Akteur initiiert und führt gewöhnlich zu einem wahrnehmbaren Ergebnis.“ (Oestereich, 2009, S. 252) Das Ergebnis kann ein Erfolg oder Fehlschlag sein. Im Use Case werden also die Benutzersicht und die Aktionen des Systems beschrieben.

OTST besteht aus zwei Programmteilen (Subsystemen), dem Administrationstool (Admin-Tool) und der eigentlichen Anwendung (Client-Tool). Die Use Cases rund um die Vorlagendefinition finden im Admin-Tool statt, die Generierung und Bearbeitung von Briefen im Client-Tool (siehe Abbildung 11, Seite 34). Je nach Use Case werden aus Anwendersicht ähnliche Use Cases (Objekt Neuerstellen, Bearbeiten, Löschen) nur im Notizteil erwähnt.

Abbildung 15 zeigt die Verwendung der Komponenten in den Use Cases. Der Viewer wird in mehreren Fällen zur Darstellung von Textblock, Vorlage und fertigem Brief verwendet. Der Generator kommt wie der Transformator nur in einem Fall zum Einsatz. Use Case A (Textbaustein definieren) verwendet keine der drei Komponenten, da er vollständig in das bestehende System integriert wird.

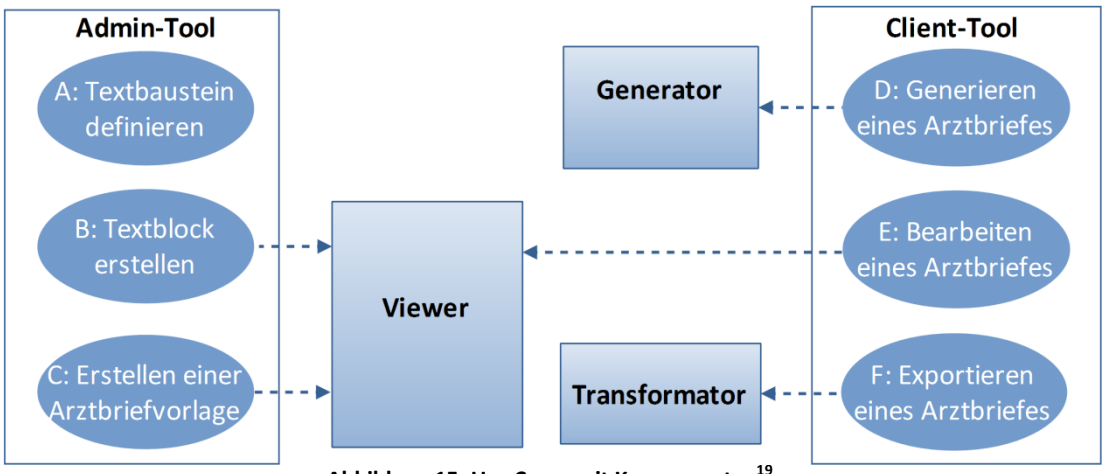


Abbildung 15: Use Cases mit Komponenten[19]

3.4.1. Use Case A: Textbaustein definieren

Subsystem:

Admin-Tool

Akteure:

Admin

Kontext und Vorbedingungen:

1. Der Benutzer ist mit Administratorberechtigung angemeldet.
2. Das Sub-Modul *Arztbriefgenerierung* ist aktiviert.

[19] Eigene Darstellung

Normalablauf:

1. Der Admin wählt im Menü eine Berichtstruktur aus.
2. Das System öffnet das Fenster zur Bearbeitung der Berichtstruktur.
3. Der Admin öffnet mit Rechtsklick auf ein Berichtselement das Kontextmenü und wählt den Menüeintrag „Textbaustein“.
4. Das System prüft auf Vorhandensein eines MCS-Codes und öffnet – falls vorhanden – ein Fenster mit einem Eingabefeld für den Textbaustein.
5. Der Admin gibt einen Textbaustein ein.
6. Der Admin wählt die Funktion „Speichern“ aus.
7. Das System speichert den Textbaustein und schließt das Fenster.
8. Der Admin schließt das Fenster der Berichtstruktur.

Normalergebnis:

Ein Textbaustein ist in der Datenbank einem Berichtselement mit MCS-Code zugeordnet.

Alternativablauf:

4a1. Das Berichtselement enthält keinen MCS-Code, der Admin erhält eine Mitteilung, die dies besagt.
4a2. Der Admin definiert im Fenster der Berichtstruktur einen MCS-Code.
Weiter mit Schritt 3.

Nicht funktionale Anforderungen:

Der Admin hat Kenntnisse über Berichtstruktur und MCS-Codes.

Notiz:

Einen Textbaustein neu zu erstellen, zu bearbeiten oder überschreiben ist aus Anwendersicht dasselbe. Es existiert intern nur ein Textbaustein pro MCS-Code.

Eine Erweiterung dieses Use Cases ist im Client-Tool vorgesehen. Dabei soll im Berichtseingabefenster der Eintrag „Textbaustein“ des Kontextmenüs ein Fenster zur Bearbeitung von Textbausteinen öffnen.

3.4.2. Use Case B: Textblock erstellen

Subsystem:

Admin-Tool

Akteure:

Admin

Kontext und Vorbedingungen:

1. Der Benutzer ist mit Administratorberechtigung angemeldet.
2. Das Sub-Modul *Arztbriefgenerierung* ist aktiviert.

Normalablauf:

1. Der Admin wählt im Menü den Eintrag „Textblöcke“.
2. Das System öffnet das Fenster zur Bearbeitung von Textblöcken. Dargestellt werden eine Liste aller vorhandenen Textblöcke mit Auswahlmöglichkeit und darunter die Eingabefelder des Textblocks: Name des Textblocks, CDA Body-Level 2 Identifier, WYSIWYG-Editor.
3. Das System aktiviert den ersten Eintrag der Liste (falls >0 vorhanden) und lädt seine Daten in die Eingabefelder.
4. Der Admin wählt den Button „Neuer Textblock“.
5. Das System setzt alle Eingabefelder auf Standard- bzw. Leerwerte.

6. Der Admin definiert den Textblock mit eindeutigem Namen und CDA Body-Level 2 Identifier.
7. Der Admin gibt im Editor einen Freitext ein.
8. Der Admin wählt den Button „Stammdaten“: Use Case B1
9. Der Admin wählt den Button „Strukturierte Daten“: Use Case B2
10. Der Admin wählt den Button „Speichern“.
11. Das System kontrolliert alle Eingabefelder und speichert den Textblock, der Admin erhält eine Mitteilung, die dies besagt.

Normalergebnis:

Ein Textblock ist in der Datenbank gespeichert.

Alternativablauf:

3a1. Der Admin wählt einen existierenden Textblocks aus.
4a2. Das System setzt alle Eingabefelder auf die Werte des gewählten Textblocks.
Weiter mit Schritt 6.

10a1. Das System erkennt einen fehlenden Wert in einem der Eingabefelder und teilt dies dem Admin mit.
10a2. Der Admin ergänzt die fehlenden.
Weiter mit Schritt 10.

Die Reihenfolge der Schritte 5 bis 8 ist flexibel.

Nicht funktionale Anforderungen:

-

Notiz:

Der Use Case „Textblock bearbeiten“ unterscheidet sich nur durch

4. Der Admin wählt einen Textblock aus der Liste.
5. Das System setzt alle Eingabefelder auf die des Textblocks.

Der Use Case „Textblock löschen“ unterscheidet sich ab Schritt 3 bis zum Ergebnis

3. Der Admin wählt einen Textblock aus der Liste.
4. Das System setzt alle Eingabefelder auf die des Textblocks.
5. Der Admin wählt den Button „Löschen“.
6. Das System verlangt eine Bestätigung und löscht alle Daten des Textblocks.
7. Das System aktiviert den ersten Eintrag der Liste (falls >0 vorhanden) und lädt seine Daten in die Eingabefelder.

Normalergebnis „Textblock löschen“:

Der Textblock ist aus der Datenbank physikalisch gelöscht.

3.4.3. Use Case B1: Einfügen eines Stammdatenausdrucks

Subsystem:

Admin-Tool

Akteure:

Admin

Kontext und Vorbedingungen:

1. Der Benutzer hat entweder das Textblock- oder das Briefvorlagen-Fenster geöffnet und den Button „Stammdaten“ gewählt.

Normalablauf:

1. Das System öffnet das Fenster zur Auswahl eines Stammdatenausdrucks, dargestellt werden Listenfelder zur Auswahl und Buttons zum Übernehmen eines Ausdrucks bzw. Abbrechen.
2. Der Admin wählt aus den Listenfeldern einen Ausdruck aus.
3. Der Admin wählt den Button „Übernehmen“.
4. Das System schließt das Fenster und fügt den gewählten Ausdruck in den Editor des übergeordneten Fensters ein.

Normalergebnis:

Ein Stammdatenausdruck wurde im Editor des übergeordneten Fensters eingefügt.

Alternativablauf:

3a1. Der Admin wählt den Button „Abbrechen“.
4a2. Das System schließt das Fenster.

Nicht funktionale Anforderungen:

-

Notiz:

-

3.4.4. Use Case B2: Einfügen eines Ausdrucks für strukturierte Daten

Subsystem:

Admin-Tool

Akteure:

Admin

Kontext und Vorbedingungen:

1. Der Benutzer hat das Textblock- oder Briefvorlagen-Fenster geöffnet und den Button „Strukturierte Daten“ gewählt.

Normalablauf:

1. Das System öffnet das Fenster zur Auswahl eines strukturierten Ausdrucks, dargestellt werden Listenfelder zur Auswahl des Berichts und zugehörgier Berichtselemente und Buttons zum Übernehmen eines Ausdrucks bzw. Abbrechen.
2. Der Admin wählt aus den Listenfeldern einen Ausdruck aus.
3. Der Admin wählt den Button „Übernehmen“.
4. Das System schließt das Fenster und fügt den gewählten Ausdruck in den Editor des übergeordneten Fensters ein.

Normalergebnis:

Ein strukturierter Ausdruck wurde im Editor des übergeordneten Fensters eingefügt.

Alternativablauf:

3a1. Der Admin wählt den Button „Abbrechen“.
4a2. Das System schließt das Fenster.

Nicht funktionale Anforderungen:

Der Benutzer hat Kenntnisse über die Struktur der Berichte.

Notiz:

-

3.4.5. Use Case C: Erstellen einer Arztbriefvorlage

Subsystem:

Admin-Tool

Akteure:

Admin

Kontext und Vorbedingungen:

1. Der Benutzer ist mit Administratorberechtigung angemeldet.
2. Das Sub-Modul *Arztbriefgenerierung* ist aktiviert.
3. In der Datenbank ist mindestens ein Textblock vorhanden.

Normalablauf:

1. Der Admin wählt im Menü den Eintrag „Briefvorlagen“.
2. Das System öffnet das Fenster zur Bearbeitung von Briefvorlagen. Dargestellt werden eine Liste aller vorhandenen Vorlagen mit Auswahlmöglichkeit, darunter die Eingabefelder der Vorlage (Name des Briefs, WYSIWYG-Editor) und eine Liste aller vorhandenen Textblöcke mit Auswahlmöglichkeit.
3. Der Admin wählt den Button „Neue Briefvorlage“.
4. Das System setzt alle Eingabefelder auf Standard- bzw. Leerwerte.
5. Der Admin definiert die Vorlage mit eindeutigem Namen.
6. Der Admin wählt einen Textblock aus und wählt den Button „Hinzufügen“.
7. Das System erstellt im Editor einen Verweis zum gewählten Textblock.
8. Der Admin gibt im Editor einen Freitext ein.
9. Der Admin wählt den Button „Stammdaten“: Use Case B1
10. Der Admin wählt den Button „Speichern“.
11. Das System kontrolliert alle Eingabefelder und speichert die Vorlage, der Admin erhält eine Mitteilung, die dies besagt.

Normalergebnis:

Eine Briefvorlage ist in der Datenbank gespeichert.

Alternativablauf:

11a1. Das System erkennt einen fehlenden Wert in einem der Eingabefelder und teilt dies dem Admin mit.
11a2. Der Admin ergänzt die fehlenden Werte.
Weiter mit Schritt 11.

Die Reihenfolge der Schritte 5 bis 9 ist flexibel.

Nicht funktionale Anforderungen:

-

Notiz:

Der Use Case „Arztbriefvorlage bearbeiten“ unterscheidet sich nur durch

3. Der Admin wählt eine Vorlage aus der Liste.
4. Das System setzt alle Eingabefelder auf die der Vorlage.

Der Use Case „Arztbriefvorlage löschen“ unterscheidet sich ab Schritt 3 bis zum Ergebnis

3. Der Admin wählt eine Vorlage aus der Liste.
4. Das System setzt alle Eingabefelder auf die der Vorlage.
5. Der Admin wählt den Button „Löschen“.
6. Das System fragt nach einer Bestätigung und löscht alle Daten der Vorlage.
7. Das System aktiviert den ersten Eintrag der Liste (falls >0 vorhanden) und lädt seine Daten in die Eingabefelder.

Normalergebnis „Arztbriefvorlage löschen“:

Die Arztbriefvorlage ist aus der Datenbank gelöscht. Falls Briefe in der Datenbank vorhanden sind, die auf der Vorlage basieren, werden die Formatierungsoptionen der Vorlage behalten, der Rest physikalisch gelöscht.

3.4.6. Use Case D: Generieren eines Arztbriefes

Subsystem:

Client-Tool

Akteure:

Arzt

Kontext und Vorbedingungen:

1. Der Benutzer verfügt über die Berechtigung zum Erstellen von Briefen.
2. Das Sub-Modul *Arztbriefgenerierung* ist aktiviert.
3. In der Datenbank ist mindestens eine Arztbriefvorlage vorhanden.

Normalablauf:

1. Der Arzt aktiviert einen Patienten.
2. Der Arzt wählt die Funktion „Briefgenerierung“
3. Das System öffnet ein Fenster zur Auswahl der Briefvorlage und Datengrundlage.
4. Der Arzt wählt eine Vorlage und als Datengrundlage eine Fall-ID (CID) des aktivierten Patienten aus.
5. Der Arzt wählt den Button „Brief generieren“.
6. Das System generiert den Arztbrief, schließt das aktuelle Fenster und öffnet das Brief-Bearbeitungsfensters: Use Case E.

Normalergebnis:

Ein Brief wurde erstellt und ist dem aktivierten Patienten mit aktuellem Datum und gewählter CID zugeordnet.

Alternativablauf:

6a1. Die Generierung des Briefes schlägt fehl, der Arzt erhält eine Mitteilung, die dies besagt.

Nicht funktionale Anforderungen:

-

Notiz:

-

3.4.7. Use Case E: Bearbeiten eines Arztbriefes

Subsystem:

Client-Tool

Akteure:

Arzt

Kontext und Vorbedingungen:

1. Der Benutzer verfügt über die Berechtigung zum Bearbeiten von Briefen.
2. Das Sub-Modul *Arztbriefgenerierung* ist aktiviert.
3. Der Benutzer hat einen Patienten aktiviert, zu dem mindestens ein Arztbrief existiert.

Normalablauf:

1. Der Arzt wählt einen Arztbrief aus.
2. Das System öffnet ein Fenster zur Bearbeitung des Arztbriefes. Dargestellt werden Eingabefelder für die allgemeinen Daten (Datum, Autor, Absender, Empfänger, Brieftyp) und ein WYSIWYG-Editor mit dem Briefinhalt (eigentlicher Text).
3. Der Arzt bearbeitet die Daten.
4. Der Arzt wählt den Button „Speichern“.
5. Das System kontrolliert alle Eingabefelder auf Korrektheit und speichert den Brief in der Datenbank.

Normalergebnis:

Die Daten des Briefes wurden in der Datenbank geändert.

Alternativablauf:

5a1. Das System erkennt bei mindestens einem Feld einen Fehler und teilt dies dem Benutzer mit.
5a2. Das System speichert in diesem Fall keine Änderungen in der Datenbank.
Weiter mit Schritt 3.

4a1. Der Arzt wählt den Button „Abbrechen“
4a2. Das System macht die Änderungen rückgängig und zeigt die alte Version des Briefes im selben Fenster an.

Nicht funktionale Anforderungen:

-

Notiz:

Der Use Case „Arztbrief löschen“ unterscheidet sich ab Schritt 3 bis zum Ergebnis

3. Der Admin wählt den Button „Löschen“.
4. Das System fragt beim Benutzer nach und löscht nach Bestätigung den Arztbrief.
5. Das System aktualisiert das Patientenfenster.

Normalergebnis „Arztbrief löschen“:

Der Arztbrief ist aus der Datenbank physikalisch gelöscht.

3.4.8. Use Case F: Exportieren eines Arztbriefes

Subsystem:

Client-Tool

Akteure:

Arzt

Kontext und Vorbedingungen:

1. Der Benutzer verfügt über die Berechtigung zum Exportieren von Briefen.
2. Das Sub-Modul *Arztbriefgenerierung* ist aktiviert.
3. Der Benutzer hat einen Arztbrief im Bearbeitungsfenster geöffnet (Use Case E)

Normalablauf:

1. Der Arzt wählt den Button „Exportieren“ aus.
2. Das System zeigt die Formatoptionen „PDF“ und „RTF“ an.
3. Der Benutzer wählt eine der Formatoptionen.
4. Das System zeigt dem Benutzer ein Fenster für die Auswahl des Speicherorts an.
5. Der Arzt wählt Ordner und Dateiname und wählt den Button „Speichern“ aus.
6. Das System exportiert den Arztbrief im gewählten Format und speichert ihn im gewählten Pfad.

Normalergebnis:

Der Arztbrief ist in einem der unterstützten Formate auf einem lokalen oder Netzwerkpfad gespeichert.

Alternativablauf:

6a1. Das Exportieren schlägt fehl, der Arzt erhält eine Mitteilung, die dies besagt.

Nicht funktionale Anforderungen:

-

Notiz:

-

3.5. Detailentwurf

Der Detailentwurf beginnt mit der Definition der Grammatik, welche die Idee der Platzhalter umsetzt. Es folgt die Betrachtung der Komponenten aus dem Blickwinkel der verwendeten Technologien und letztendlich die Abgrenzung der Komponenten untereinander bzw. ihr Zusammenspiel miteinander in der Definition der Schnittstellen.

3.5.1. Grammatik

Die Grammatik definiert die Platzhalter, welche in den Briefvorlagen in den Text eingebunden werden. Um sie im Vorlagentext technisch einfach zu finden, werden sie per Definition in „#“-Zeichen eingeschlossen. Die Elemente eines Ausdrucks werden mit „.“-Zeichen voneinander getrennt.

Abbildung 16 zeigt die Struktur der Grammatik. Der oberste Grammatik-Ausdruck `Expression` geht auf in den Ausdruck `Master-Data` für Stammdaten und den Ausdruck `Report-Data` für die Berichtsdaten.

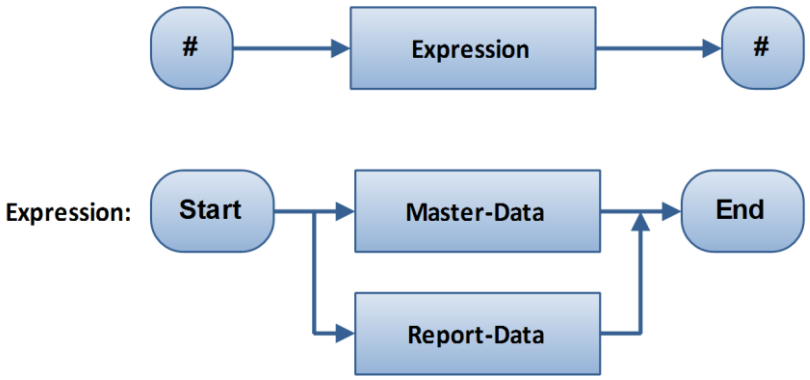


Abbildung 16: Grammatik[20]

Abbildung 17 zeigt die Auflösung des `Master-Data` Ausdrucks. Er enthält als erstes Element die Datenquelle und als zweites den Inhalt. Für die Datenquelle `Patient` sind Werte für Name, Adressdaten und Geburtsdatum vorgesehen (in der Grafik nicht alle dargestellt). Für die Datenquelle `Surgeon` (Chirurg, Arzt) nur Namen und Titel.

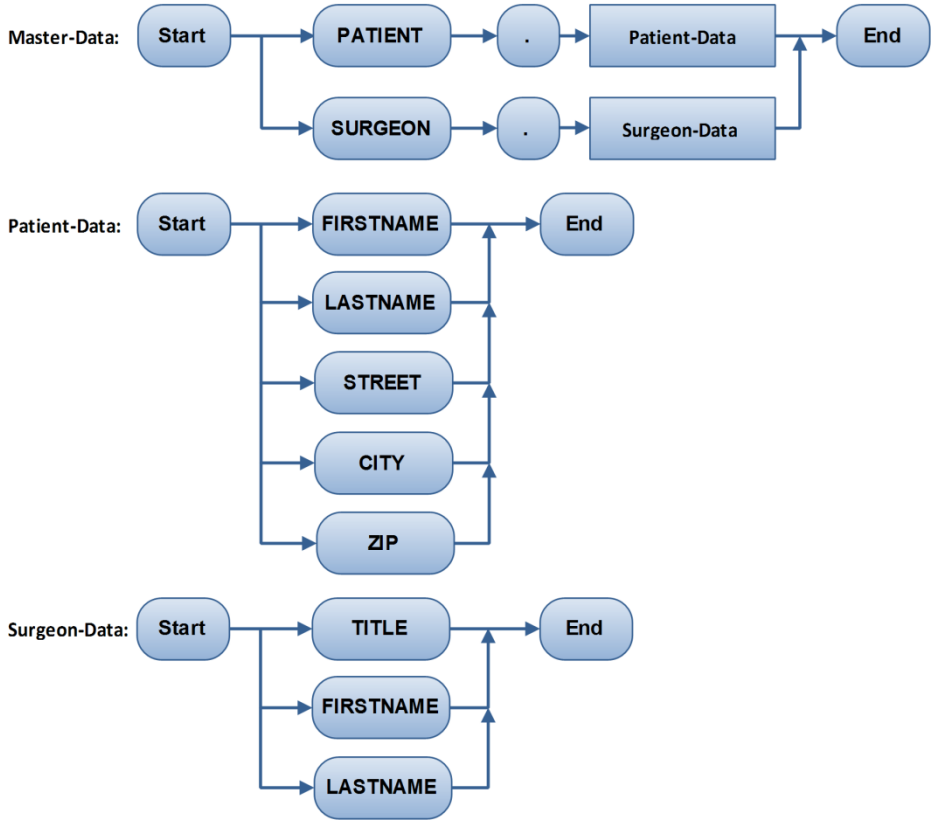


Abbildung 17: Grammatik, Master-Data Ausdrücke[21]

Abbildung 18 zeigt die Auflösung der `Report-Data` Ausdrücke. Er beginnt zur einfachen Unterscheidung vom `Master-Data` Ausdruck mit dem Bezeichner `MCS` und wird aufgeteilt in ein Element `Format` und sich optional wiederholende `Report-Content` Elemente. Letzteres besteht aus einem Bezeichner für den Wert (`Content`), dem `MCS-Code` als eindeutige Referenz zum

[20] Eigene Darstellung
[21] Eigene Darstellung

Berichtselement und dem `Children`-Element. Der Children-Zahlenwert definiert, wie viele Kind-Ebenen des referenzierten Elements verarbeitet werden sollen.

Beim Ausdruck `Format` steht `ENUM` für die Aufzählung in Fließtextform, `LISTSORT` bzw. `LISTUNSO` für Aufzählung als (nicht)nummerierte Liste. Der `Content`-Ausdruck teilt auf sich in `NAME` für den Namen, `VALUE` für den Wert und `DESCR` für den Textbaustein des Elements. Der Ausdruck `Children` kann Werte von 0 bis 4 annehmen, da die Berichtstrukturen über maximal fünf Ebenen verfügen. Die `MCS-Codes` sind in der Datenbank gespeichert und in der Grammatik nicht fest definiert, ein Code kann ein beliebiger String sein.

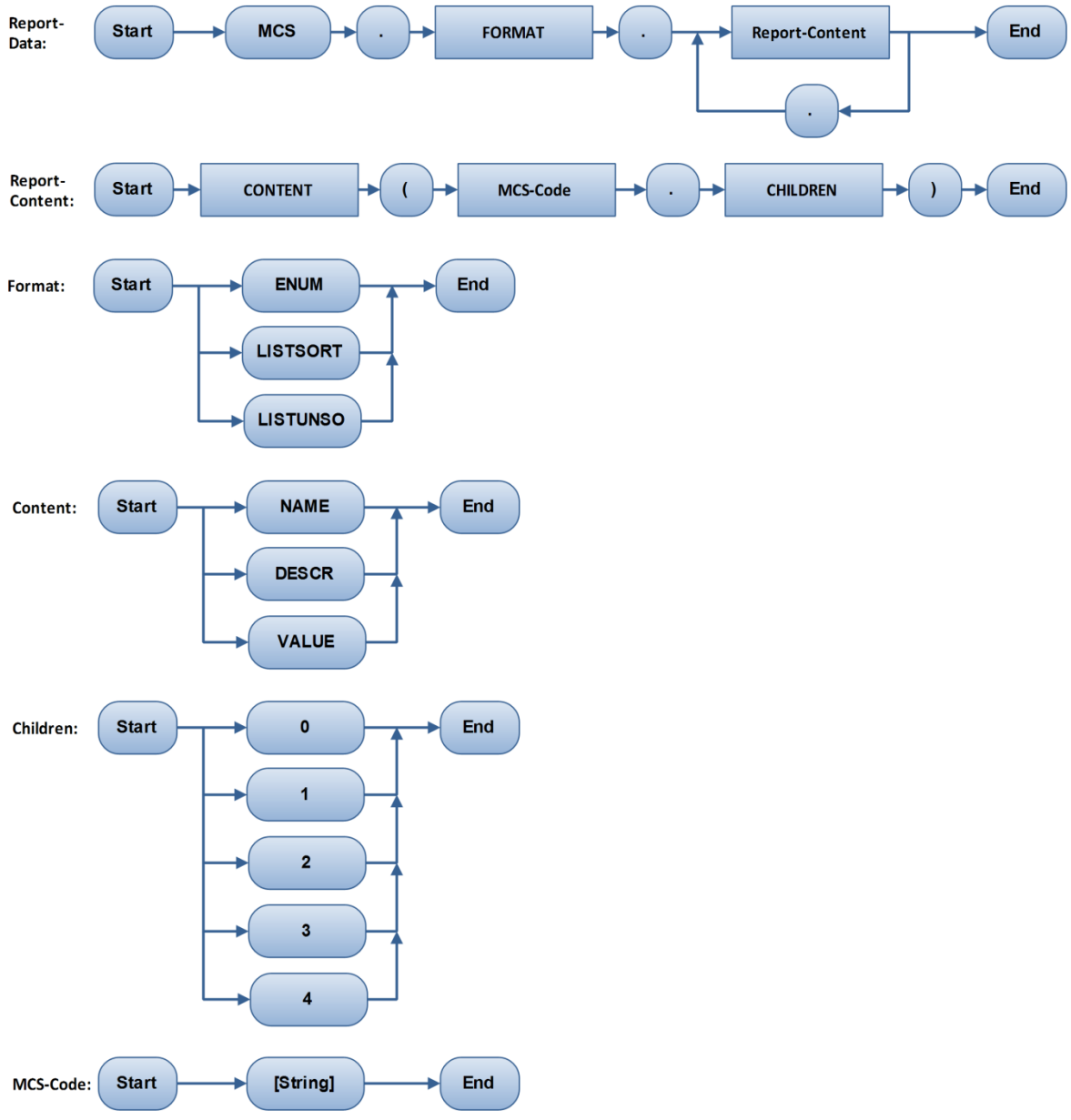


Abbildung 18: Grammatik, Report-Data Ausdrücke[22]

[22] Eigene Darstellung

Beispiel: Stammdatenausdruck

In der Briefvorlage wird der folgende Text definiert:

```
Ich berichte Ihnen über den Patienten #PATIENT.FIRSTNAME#
#PATIENT.SURNAME#, geboren am #PATIENT.BIRTHDATE#.
```

Bei der Generierung des Briefes werden die mit „#“ eingeschlossenen Ausdrücke aufgelöst, das Resultat ist folgendes:

```
Ich berichte Ihnen über den Patienten Max Muster, geboren am 01.01.1970.
```

Beispiel: Berichtsdaten-Ausdruck

In der Briefvorlage wird der folgende Text definiert. „THER“ ist dabei beispielhaft der MCS-Code des Baumknotens eines Operationsberichts, der als Kindelemente die Therapien enthält:

```
#MCS.LISTSORT.DESCR(THER.4)#
```

Die Auflösung ergibt bei angenommen zwei aktivierten Elementen des Beispiels aus dem Kapitel 2.1.2 - Flexible Berichte folgenden Text:

```
Bei der Operation wurden die folgenden Therapien durchgeführt:
  1. Paukendrainage mit Goldröhrchen
  2. Ossikuloplastik mit Titanstapes der Länge 5mm
```

Der nächste Schritt ist die weitere Verfeinerung der Komponenten. Davor werden Entscheidungen getroffen, welche die Implementierung grundsätzlich befolgen muss.

3.5.2. Technologien

Die Softwarearchitektur bzw. die Aufteilung in Komponenten legt noch nicht fest, wie diese intern technisch realisiert werden. Dies betrifft den Dokumentenstandard und die angestrebte Lösung zur Textgenerierung und wird an dieser Stelle beschrieben.

Dokumentenstandard

Aufgrund aktueller Projekte von HL7 Schweiz, der Mitgliedschaft von großer Krankenhäuser im Verein HL7 Schweiz und internationaler Verwendung des HL7-CDA Standards sollen die erstellen Dokumente in einem CDA-CH-II konformen Format abgespeichert werden. Damit besteht die Möglichkeit, die Dokumente ohne zusätzlich zu implementierende Konvertierungslogik zu einem späteren Zeitpunkt direkt jenen Systemen zur Verfügung zu stellen, die sich um den Datenaustausch mit externen Organisationen kümmern.

Um Zugriff auf alle Spezifikationen zu erhalten wird der Firma der Beitritt zur „HL7 Benutzergruppe Schweiz“ empfohlen.

Textgenerierung

Für die Generierung von Text aus der bestehenden OTST-Datenbank wird eine Eigenentwicklung angestrebt, da das Modul vollständig in das bestehende Delphi-Programm eingebunden werden soll. Die analysierten Reporting-Tools könnten einerseits Lizenzprobleme mit sich bringen und andererseits bieten sie keine direkte Möglichkeit, die Dokumente in einer für das Gesundheitswesen standardisierten Form abzuspeichern.

Editor

Der Editor soll grundlegende Formatierungsmöglichkeiten (fett, kursiv, Tabelle, Liste) bieten und diese für den Anwender darstellen. Es soll ein WYSIWYG-Editor („What you see is what you get“) entwickelt bzw. eine bestehende Delphi-Komponente verwendet und ggfs. erweitert werden. Die mit Sourcecode erhältliche, kommerzielle Komponente „ProfDHTMLEdit“ (ProfGrid.com) bietet hier die meisten Möglichkeiten (siehe Kapitel 2.5.5).

Technologieauswahl für Komponenten

Abbildung 19 zeigt – wie Abbildung 14 auf Seite 37 – noch einmal die Komponenten mit Eingabe und Ausgabe, ergänzt um die Technologie (in fetter Schrift), auf welche die Implementierung hauptsächlich aufbaut.

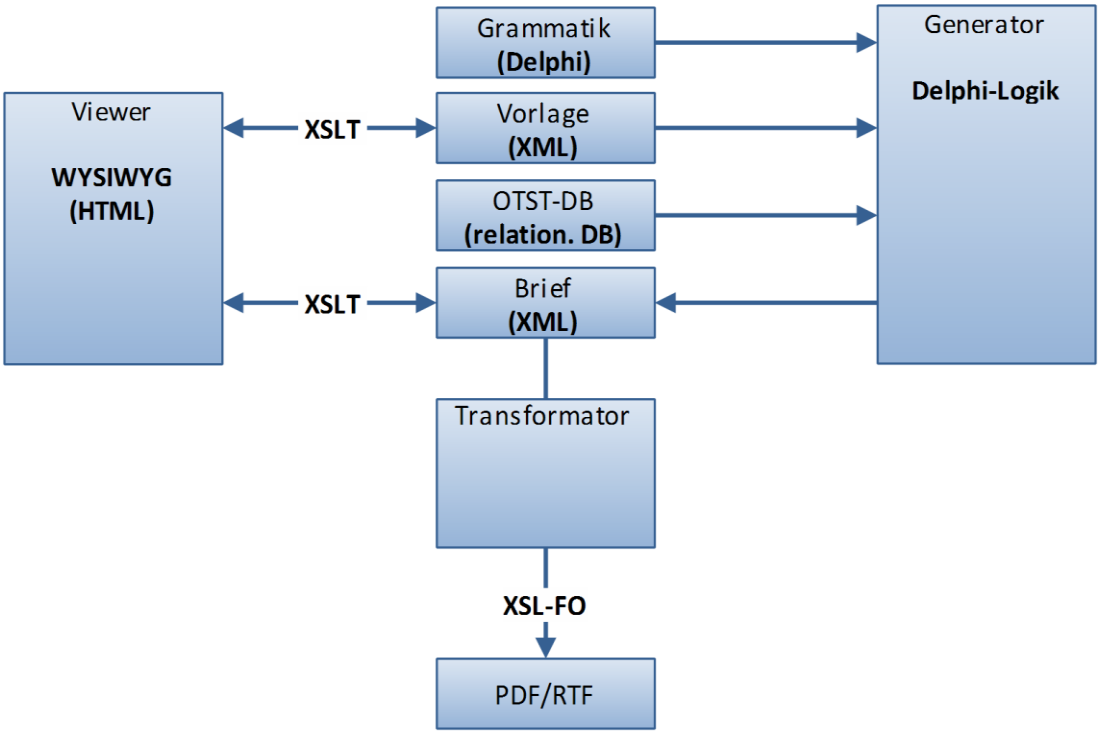


Abbildung 19: Komponenten mit Technologien[23]

Technisch überschneiden sich die Aufgaben der funktionalen Komponenten teilweise. So wird mit der Verwendung einer HTML-WYSIWYG-Komponente eine Transformation der internen Darstellung in HTML notwendig. Für diesen Zweck kann jedoch eine in Delphi vorhandene XSLT-Komponente genutzt werden. Die Transformation in die Formate PDF und RTF sind damit nicht zu bewerkstelligen.

Die Grammatik wird vom Generator verwendet. Er erstellt aus der Vorlage den Brief im CDA-Format. Dazu soll eine eigene CDA-Klasse geschaffen werden, die auch vom Transformator verwendet werden kann.

Der Viewer stellt sowohl die Vorlage als auch den Brief dar (bzw. Teile davon). Speichert man die Vorlage als XML, dessen Elemente den CDA-Elementen entsprechen (größtenteils, nicht vollständig), fallen für den Generator einige Transformationen weg und für den Viewer fällt ein zusätzliches Eingabeformat weg.

[23] Eigene Darstellung

3.5.3. Schnittstellen

Um in der Lage zu sein, die Komponenten gegebenenfalls wieder zu verwenden oder auszutauschen und von den anderen genannten Vorteile der Aufteilung profitieren zu können, müssen die Schnittstellen der Komponenten klar definiert werden.

Es soll eine Viewer-Klasse entwickelt werden, die in drei Use Cases und damit drei verschiedenen Formularen verwendet werden kann (Use Cases B, C und E). Sie muss jeweils nur das Body-Element eines CDA-Dokuments im WYSIWYG-Editor darstellen. Intern wandelt sie das Body-Element in HTML um, also kann als Eingabe entweder das gesamte CDA-Dokument als XML oder nur das Body-Element übergeben werden. Implementiert man die Viewer-Klasse so, dass das gesamte CDA-Dokument übergeben werden kann, hat man damit einen simplen CDA-Body-Viewer erstellt.

Nach der Bearbeitung durch den Anwender setzt das System das geänderte Body-Element in das CDA-Dokument ein. Als getter-Methode muss der Viewer also ein XML-Dokument mit einem validen CDA-Body liefern, der Rest des CDA-Dokuments wird vom Parent-Formular editiert. In den Use Cases kann der Benutzer Grammatik-Ausdrücke aus anderen Fenstern einfügen. Also benötigt die Klasse eine weiter öffentliche Methode zum Einfügen von einfachen Strings.

Pseudocode Viewer-Klasse:

```
Class Viewer
  public function showCDA(xml)                  : void
  public function getCDA()                      : xml
  public function addStringAtCursorPos(string)  : void
```

Der Generator erstellt aus der Vorlage ein CDA-Dokument. Die Vorlage enthält Ausdrücke, die durch die Grammatik definiert werden. Die Auflösung der Ausdrücke (Ersetzen von Platzhaltern mit realen Werten) wird vom Generator übernommen, sie ist von der Grammatikdefinition abhängig. Bei der Auflösung wird als zweite Information die Datengrundlage benötigt. Diese kann entweder die gesamte Behandlungsgeschichte eines Patienten oder nur ein Teil davon sein. Für mehr Flexibilität soll deshalb eine Zwischenebene eingeschaltet werden, die ein oder mehrere Behandlungselemente enthalten kann.

Pseudocode Generator-Klassen:

```
Class Generator
  public function resolveTemplate(xml, datasource) : xml
Class Grammar
  static fields
```

Der Transformator muss aus einem CDA-Dokument ein PDF- oder RTF-Dokument erstellen. Im CDA-Standard sind dafür jedoch nicht genügend Visualisierungselemente vorgesehen, also braucht es zusätzlich zum CDA-Format folgende Formatierungsdefinitionen:

- Seitengröße und Seitenränder
- Kopf- und Fußzeilen
- Schriftart, Schriftgröße

Diese können in der Briefvorlage hinterlegt werden, die CDA-ähnlich als XML gespeichert wird. Es kann also entweder ein zusätzliches Element mit Formatierungsoptionen definiert werden oder diese gesondert gespeichert werden. Die Transformation in PDF kann technisch mit XSL:FO realisiert werden. Dabei wird ein XML-Dokument per XSLT Transformation in ein XSL:FO Dokument gebracht (auch XML Format) und danach das PDF generiert. Die Formatierungsoptionen können also in einem eigenen XML-Dokument hinterlegt werden, das vom Transformator genutzt wird.

Alternativ können die Formatierungsangaben in verschiedenen Spalten der relationalen Datenbank abgelegt werden und der Transformator erstellt sein Transformationsdokument daraus selbst.

Pseudocode Transformator-Klasse:

```
Class Transformator
  /* First  parameter letter in xml-format
     Second parameter format-definitions in xml-format */
  public function transform2Pdf(xml, xml, filename) : void
  public function transform2Rtf(xml, xml, filename) : void
```

Zur Sicherstellung der Validität der erstellten CDA-Dokumente wird eine weitere Klasse, die CDA-Klasse benötigt. Sie muss mindestens eine Methode mit den Parametern XML-Dokument und XML-Schema zur Validierung zur Verfügung stellen. Da die Use Cases B, C und E Teile des CDA-Dokuments im Viewer und Teile in eigenen Eingabefeldern darstellen, sollte sie auch Methoden zum einfachen Auslesen und Setzen der CDA-Header-Elemente bieten.

Pseudocode CDA-Klassen:

```
Class CDAClass
  /* Parameter is letter in cda-format */
  public function setXML(xml)           : void
  /* Parameter is XML-schema */
  public function validateXML(xml)      : boolean
  public function setPatient(CDAPerson) : void
  public function getPatient()          : CDAPerson
  public function setAuthor(CDAPerson)  : void
  public function getAuthor()           : CDAPerson
  ...
Class CDAPerson
Class CDAContact
Class CDAOrganisation
```

Die Header-Elemente des CDA-Standards gleichen sich, sie enthalten beispielsweise gleich strukturierte Personen- oder Kontaktdaten. Um die Implementierung zu vereinfachen ist hier vorgesehen, mehrere Parameter in einfache Objekte zusammenzufassen, z.B. CDAPerson, CDAContact, CDAOrganisation.

4. Umsetzung

Dieses Kapitel beschreibt die Umsetzung der Vorgaben aus dem Kapitel Anforderungen und Entwurf. Im ersten Abschnitt *Übersicht* werden alle funktionalen Klassen mit ihren Beziehungen beschrieben, außerdem ihre Verwendung in den Formularen, die die Use Cases umsetzen.

Die restlichen Abschnitte beschreiben die Klassen mit öffentlichen Methoden, interner Realisierung und mit Erklärung von benötigten Tabellen. Diese Klassen können gegebenenfalls auch in einer anderen Entwicklung wieder verwendet werden. Der Quellcode wird aus rechtlichen Gründen nicht vollständig abgebildet, er ist Eigentum der Firma *innoForce Est.* Die vollständigen Erklärungen zu allen implementieren Formularen sind aus Platzgründen nicht abgebildet.

Wo sinnvoll, sind Schlüsselstellen mit Code-Abschnitten erklärt, ansonsten werden ausschließlich die Methoden mit Parametern aufgeführt und anhand von Sequenzdiagrammen und im Text erklärt.

4.1. Übersicht

Die Logik des Moduls wird in 6 Hauptklassen unterteilt. Der Viewer wird im Fenster `frmCDAEditor` realisiert. Der Zugriff auf XML-Daten und damit auch XSL-Stylesheets erfolgt zentral über die Hilfsklasse `CoHelper` zur Tabelle `TU_XML_Data`. Der Generator ist in der Klasse `LtrGenerator` realisiert. Er enthält verschiedene Datenbankzugriffe auf die gesamte OTST-Datenbank, verwendet für das Einlesen der XML-Briefvorlagen und Speichern von XML-Briefen die Hilfsklasse, zum Erstellen des Briefes die Klasse `CDADocument`. `CDADocument` ist die Klasse zur Erstellung von CDA-konformen XML-Dokumenten. Sie hält den Inhalt des CDA-Dokuments in von `CDAElement` erbenden Instanzen, in der Grafik dargestellt am Beispiel Patient in der Klasse `CDAPatient`. Für das Auflösen der Grammatik-Ausdrücke aus der Briefvorlage verwendet der Generator Definitionen aus der Grammatik. Diese wird in der Klasse `LtrGrammar` und der Tabelle `TS_GRA_Grammar` definiert. Die Transformationsklasse `LtrTransformator` liest XML-Briefe und Formatangaben über die Hilfsklasse ein.

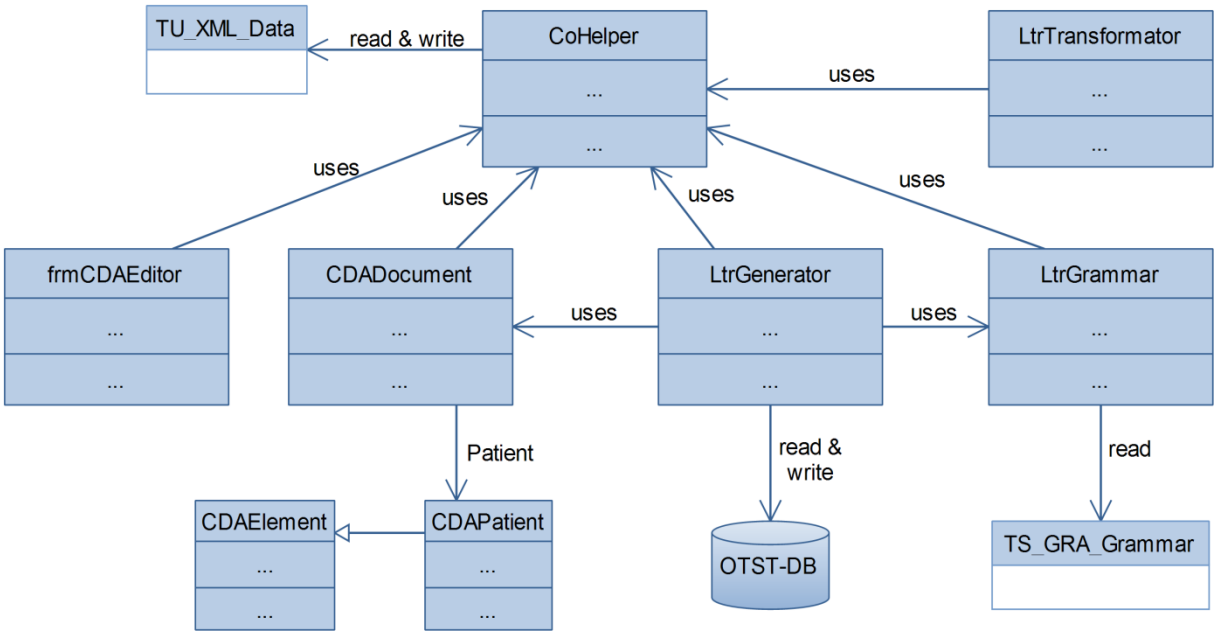


Abbildung 20: Übersicht Klassen, Beziehungen, Datenbank

Die aufgeführten Klassen werden in den Formularen verwendet, welche die Use Cases implementieren. In den Formularen für *Use Case B: Textblock erstellen* und *Use Case C: Erstellen einer Arztbriefvorlage* werden das Viewer-Frame zur Darstellung und die Hilfsklasse zum Einlesen der als XML gespeicherten Vorlage bzw. Blocks verwendet. Für *Use Case E: Bearbeiten eines Arztbriefes* kommt zusätzlich die Klasse `CDADocument` für die CDA-konforme Speicherung zum Einsatz. Die Grammatikklasse wird von *Use Case B1: Einfügen eines Stammdatenausdrucks* und *Use Case B2:*

Einfügen eines Ausdrucks für strukturierte Daten verwendet, außerdem von der Generatorklasse. Letztere kommt im *Use Case D: Generieren eines Arztbriefes* zum Einsatz. Die Transformatorklasse wird in *Use Case F: Exportieren eines Arztbriefes* verwendet.

4.2. Hilfsklasse

Die Hilfsklasse `CoHelper` (Communication-Hilfsklasse) ist zuständig für das Ein- und Auslesen von XML-Dokumenten und stellt außerdem einfache Hilfsfunktionen zur Verfügung, die an mehreren Stellen im Programm benötigt werden.

4.2.1. XML-Speicherung

XML als Dokumentenformat wird an mehreren Stellen verwendet: Zur Speicherung von Textblöcken, Briefvorlagen und des Briefes selbst. Da die Software OTST sämtliche Daten in einer relationalen Datenbank abspeichert, sollen auch die XML-Dokumente selbst in der Datenbank abgelegt werden. Für das Modul *Arztbriefgenerierung* wird ein einzelner Speicherort für alle XML-Dokumente angestrebt, wobei das Ein- und Auslesen über zentrale Methoden erfolgen soll. Dies bringt den Vorteil, dass die Art der Speicherung umgebaut werden kann und Änderungen nur an einer Stelle erfolgen müssen.

Die Tabelle `TU_XML_Data` legt das gesamte XML-Dokument als String in einem Textfeld ab. Dies hat den Vorteil, dass es einfach auszulesen ist und im Delphi-Code nicht umgewandelt werden muss. Die Verwendung eines einfachen Textfeldes statt des ab MS SQL Server 2005 verfügbaren XML-Datentyps hat den Vorteil, dass die Datenbank auch mit SQL Server 2000 verwendet werden kann, was der Testumgebung in der Firma entspricht.

Tabelle 3: Tabelle TU_XML_Data

Zelle	Datentyp	Größe [Bit]	Erklärung
U_XML_ID	Integer	4	Primärschlüssel, Auto-Value
U_XML_String	Text	16	Enthält das XML-Dokument als String

Für das Ein- und Auslesen von XML-Daten aus der Datenbank wird eine `read`- und `write`-Methode implementiert, die ein XML-Dokument als String akzeptiert. Als Parameter erwartet die `read`-Methode die ID des Tabelleneintrags. Der `write`-Methode wird bei Neuerstellung eines Eintrags als ID im ersten Parameter eine ungültige ID (-1) übergeben, sie liefert die ID des erstellten Eintrags als Rückgabewert zurück. Für das Bearbeiten bzw. Überschreiben eines Eintrags wird die aktuelle ID im ersten Argument übergeben und derselbe Wert zurückgeliefert.

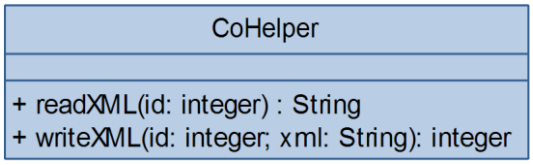


Abbildung 21: Hilfsklasse mit XML-Lese- & Schreibmethoden

Die Implementierung der Methoden ist trivial, sie verwendet sowohl zum Ein- als auch zum Auslesen einfache SQL-Abfragen.

4.2.2. Hilfsfunktionen

Für das Bearbeiten von XML-Dokumenten in Delphi wird die Delphi-XML-Klasse `TNativeXML` verwendet. Mit ihr kann ein XML-Dokument nach dem vom DOM-Standard definierten Prinzip bearbeitet werden. Sie bietet damit Methoden zum Erstellen, Finden und Löschen von Elementen, Attributen und Namensräumen. Die Transformation von String in `TNativeXML` geschieht in den

zwei `load`-Methoden der Hilfsklasse. Der jeweils zweite Parameter ist als Call-by-Reference-Parameter realisiert und dient als Rückgabewert.

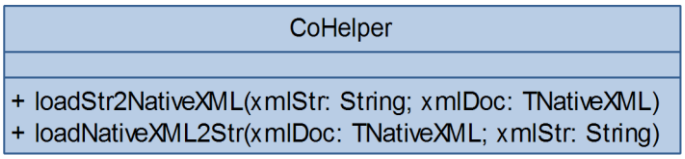


Abbildung 22: Hilfsklasse mit XML-Transformationsmethoden

4.3. Viewer

Der Viewer basiert auf der bestehenden Delphi-Komponente `ProfDHTMLEdit` (WYSIWYG-Editor, siehe Kapitel 2.5.5) und einem XSLT-Prozessor. Die Editor-Komponente verlangt als Eingabe HTML-Code, bietet Methoden zum einfachen Formatieren und Erstellen von Textstrukturen (Listen, Tabellen) und liefert HTML-Code zurück. Für die Darstellung der in XML gespeicherten Textblöcke, Briefvorlagen und Briefe müssen diese also per XSLT-Transformation in HTML transformiert werden.

Abbildung 23 zeigt das Prinzip der Transformation: Der CDA-Brief (XML-Format) wird mit Hilfe eines Stylesheets (CDA2HTML) vom XSLT-Prozessor in HTML umgewandelt. Der WYSIWYG-Editor stellt den HTML-Code dar und liefert nach Änderungen durch den Benutzer wiederum HTML-Code zurück. Der XSLT-Prozessor wandelt diesen mit Hilfe eines anderen Stylesheets (HTML2CDA) in XML um und liefert so das Body-Element des CDA-Dokuments.

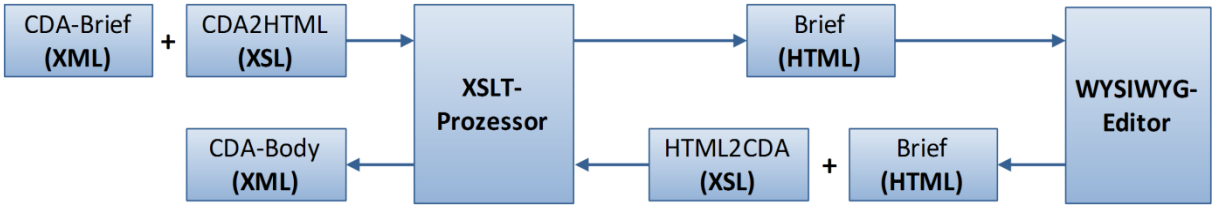


Abbildung 23: Editor-Prinzip[24]

Der Viewer wird so entworfen, dass er ausschließlich den Body einer HL7-CDA Datei darstellt. Der Rest der XML-Datei wird von der Viewer-Klasse zwar eingelesen, jedoch nicht verändert und kann je nach Verwendung des Viewers (in mehreren Use Cases, also verschiedenen Formularen) extra dargestellt und manipuliert werden.

Es wird eine Klasse zur Darstellung eines Fensters (Frame) implementiert, die die Komponente `ProfDHTMLEdit` enthält und mehrere öffentliche Methoden zum Ein- und Auslesen zur Verfügung stellt. Abbildung 24 zeigt die Methoden mit Parametern. Die ersten zwei Methoden (`show`) stellen den Body eines CDA-Dokuments im WYSIWYG-Editor dar. Die `append`-Methoden hängen den Body eines CDA-Dokuments an die aktuelle Cursorposition im Editor an. Die Methode `appendSimpleString` fügt einen einfachen String ein. Die `get`-Methoden liefern ein CDA-Dokument mit dem im Editor dargestellten Body zurück, wobei die restlichen XML-Elemente die des anfangs in `showCDAXML` übergebenen Dokuments sind. Die `get`-Methoden liefern den Rückgabewert im übergebenen Parameter (Prinzip Call-by-Reference) zurück. Für die `show`-, `get`- und `append`-Methoden werden jeweils zwei verschiedene Parameter angeboten: String und der Delphi-XML-Datentyp `TNativeXML`.

[24] Eigene Darstellung

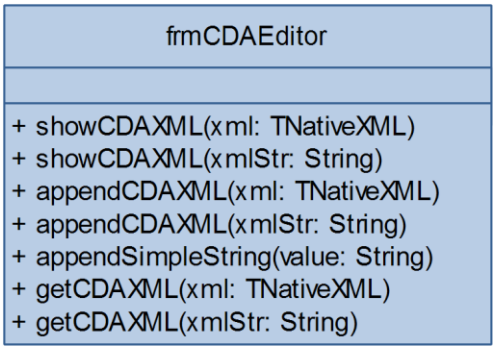


Abbildung 24: Editor-Frame mit öffentlichen Methoden

Intern muss die Viewer-Klasse zwei Transformationen durchführen, von CDA nach HTML und umgekehrt, was in zwei privaten Methoden geschieht. Das mit dem ersten `show`-Aufruf erhaltene CDA-Dokument wird als DOM-Baum zwischengespeichert (`xmlDoc`).

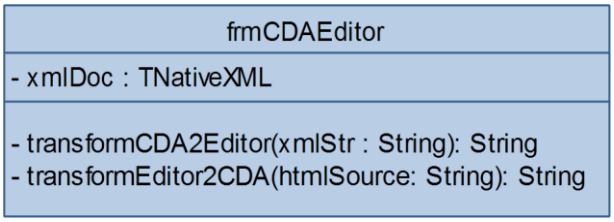


Abbildung 25: Editor-Frame mit privaten Variablen und Methoden

Für das Durchführen der XSLT-Transformation in Delphi wird die Delphi-Komponente `TXSLPageProducer` verwendet. Einziges Problem dabei ist, dass dessen Methoden nicht einheitliche Ein- und Ausgabeformate verlangen (`String`, `TStringList`, `TXMLDocument`). `TXMLDocument` sieht ein XML-Dokument auch als DOM-Baum, bietet jedoch weniger Möglichkeiten als die sonst verwendete Klasse `TNativeXML`. Eine Instanz von `TXMLDocument` kann mit einem einfachen String initialisiert werden.

Die Transformation eines XML-Dokuments mit einem XSL-Stylesheet unter Verwendung der Komponente `TXSLPageProducer` ist am Beispiel der Transformation HTML nach CDA im folgenden Code-Abschnitt dargestellt. Die Komponente verlangt das zu transformierende XML-Dokument als Instanz von `TXMLDocument`, das Stylesheet als `TStringList` und liefert das transformierte XML als `String` zurück. Das Stylesheet wird an anderer Stelle aus der Datenbank in die `TStringList` geladen.

```
function TfrmCDAEditor.transformEditor2CDA(htmlSource: WideString): String;
var
  XMLDocument : TXMLDocument;
  xslList     : TStringList;
  xslt        : TXSLPageProducer;
begin
  // Preparation of Variables
  XMLDocument := TXMLDocument.Create(Self);
  XMLDocument.Active := false;
  XMLDocument.LoadFromXML(htmlSource);
  xslt := TXSLPageProducer.Create();
  xslt.XMLData := XMLDocument;                    // 1. XML-Input
  XMLDocument.Active := true;

  // set XSL-Stylesheet
  xslt.XML := xslEditor2CDA;                      // 2. Stylesheet

  // Perform Transformation
  Result := xsl1.Content;                         // 3. Result is Return-Value
  XMLDocument.Active := false;
```

```
  XMLDocument.Free
  xslt.Free
end;
```

Das Stylesheet zur Transformation von HL7-CDA nach HTML kann in drei Abschnitten erklärt werden, Teile davon sind hier abgebildet. Ein HTML-Dokument wird über das Stylesheet erstellt, sofern ein CDA-Body-Element vorhanden ist. Wichtig dabei ist die vollständige Angabe des Namensraums. Im Body-Element wird für jedes `component`-Element das `text`-Element bearbeitet. Ein `div`-Element wird eingefügt, um in der Viewer-Klasse die Unterteilung des Bodys zu erhalten.

```
<xsl:template match="/">
  <html><head></head><body>
  <xsl:apply-templates select=
       "cda:ClinicalDocument/cda:component/cda:structuredBody"/>
  </body></html>
</xsl:template>

<xsl:template match=
      "cda:ClinicalDocument/cda:component/cda:structuredBody">
  <xsl:for-each select="cda:component">
    <xsl:apply-templates select="cda:section/cda:text"/>
  </xsl:for-each>
</xsl:template>

<!-- special div-tag for Editor-component -->
<xsl:template match="cda:section/cda:text" name="cda_section_text">
      <div>
            <xsl:call-template name="cda_text"/>
      </div>
</xsl:template>

<xsl:template match="cda:text" name="cda_text">
      <xsl:apply-templates/>
</xsl:template>
```

Die Transformation einer HL7-CDA-Liste (nicht nummeriert) in HTML erfolgt im folgenden Abschnitt:

```
<xsl:template match="cda:list">
  <xsl:if test="@listType='unsorted'"><ul>
    <xsl:apply-templates select="node()"/></ul>
  </xsl:if>
</xsl:template>
<xsl:template match="cda:item">
      <li><xsl:apply-templates select="node()"/></li>
</xsl:template>
```

Der HL7-CDA-Standard kennzeichnet Textformatierung (z.B. fett) im Attribut, nicht als Element wie im HTML-Standard, deshalb ist eine weitere Transformation notwendig:

```
<xsl:template match="cda:content">
  <xsl:if test="@styleCode='Bold'">
    <b><xsl:apply-templates select="node()"/></b>
  </xsl:if>
</xsl:template>
```

4.4. CDA-Klasse

Die CDA-Klasse wird dazu verwendet, ein XML-Dokument einfach mit den Elementen zu vervollständigen, die vom HL7-CDA Standard verlangt werden. Abbildung 26 zeigt die öffentlichen Methoden der Klasse. Die Methode `addFullBodyXML` wird zur Initiierung mit dem von der Viewer-Klasse gelieferten XML-String verwendet. Durch Aufrufen der `set`-Methoden (`setPatient`, `setAuthor` etc.) können die Header-Elemente hinzugefügt werden. Die `generateCDA`-Methode liefert einen String zurück, der das gesamte CDA-Dokument im XML-Format enthält.

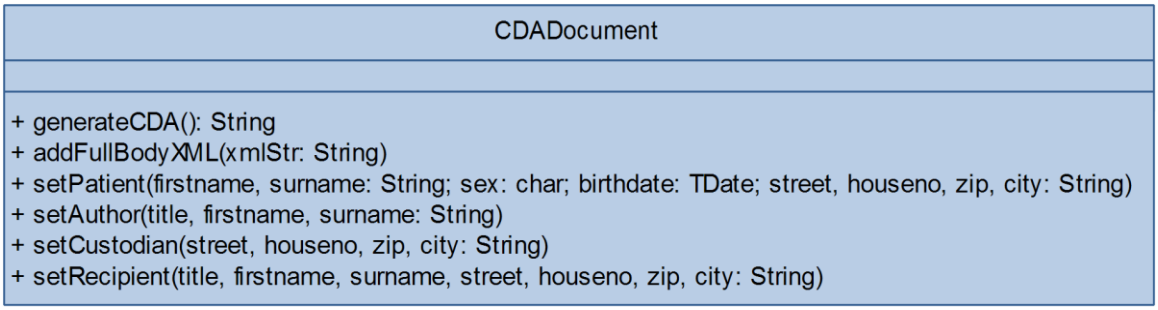


Abbildung 26: CDA-Klasse mit öffentlichen Methoden

Intern hält die Klasse das CDA-Dokument in einer Instanz von `TNativeXML`. Die Daten von Patient, Autor usw. sind in privaten Variablen abgelegt, deren Datentypen von der Klasse `CDAElement` abgeleitet sind. Diese Klasse verfügt über eine abstrakte Methode `element2XML`, die von den ableitenden Klassen so implementiert wird, dass sie einen XML-Knoten vom Typ `TXMLNode` (Knotentyp von `TNativeXML`) zurückliefern. Dieser kann von der Klasse CDADocument einfach in die `TNativeXML`-Instanz eingesetzt werden.

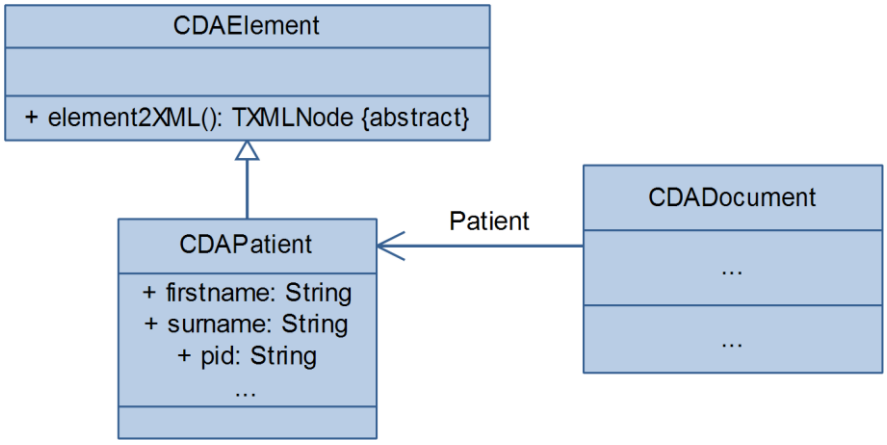


Abbildung 27: Klassendiagramm CDAElement, CDADocument

4.5. Grammatik

Die Grammatik wird in einer Delphi Unit definiert. Sie wird von zwei Seiten verwendet: In den Formularen zur Bearbeitung von Blöcken bzw. Vorlagen (Generierung von Grammatik-Ausdrücken), vom Generator bei der Generierung von Text (Parsen von Grammatik-Ausdrücken). Nach dem Parsen eines Ausdrucks erstellt der Generator daraus Text. Eine Fehlerbehandlung für fehlerhafte Ausdrücke ist nicht notwendig, die Formulare rund um den Viewer sind dafür verantwortlich, dass nur korrekte Ausdrücke in die Vorlagen eingefügt werden, fehlerhafte Ausdrücke können also durch Leertext ersetzt werden.

Die Grammatik-Unit besteht im Grunde nur aus String-Konstanten, die von anderen Klassen verwendet werden. Die Korrektheit der Ausrücke wird von der Parse-Methode des Generator

überprüft (siehe Kapitel 4.6). Die Definition der Ausdrücke teilt sich in die zwei Bereiche Stammdaten und Berichtsdaten auf.

4.5.1. Stammdaten

Ein Stammdaten-Ausdruck wird unterteilt in zwei Teilausdrücke (siehe Grammatikdefinition im Kapitel 3.5.1). Die Teilausdrücke werden in der Tabelle `TS_GRA_Grammar` hinterlegt. Tabelle 4 zeigt die Spalten der Tabelle. Neben dem Code in `S_GRA_Code` (z.B. PATIENT, FIRSTNAME) enthält jeder Datensatz Informationen zur Abfolge, die bei der Anzeige verwendet werden können (`S_GRA_ListBoxNo`, `S_GRA_ChildBoxNo`). Eine weitere Spalte (`S_GRA_Caption`) dient der Beschriftung, dadurch können die Ausdrücke je nach aktivierter Sprache im Programm anders angezeigt werden.

Tabelle 4: Tabelle TS_GRA_Grammar

Spalte (Zelle)	Datentyp	Größe [Bit]	Erklärung
S_GRA_ID	Integer	4	Primärschlüssel, Auto-Value
S_GRA_Code	Varchar	10	Code, z.B. PATIENT, FIRSTNAME
S_GRA_ListBoxNo	Integer	4	Identifier der Ebene
S_GRA_ChildBoxNo	Integer	4	Identifier, welche Ebene darauf folgt
S_GRA_Caption	Varchar	50	Beschriftung für den Anwender

Anwendersicht

Der Anwender sieht bei der Auswahl eines Stammdatenausdrucks zwei Listenfelder, die aufgrund der Tabelleneinträge gefüllt werden (siehe Abbildung 28). Angezeigt werden die Beschriftungen aus `S_GRA_Caption`, im unteren Teil wird der daraus resultierende Grammatik-Ausdruck angezeigt.

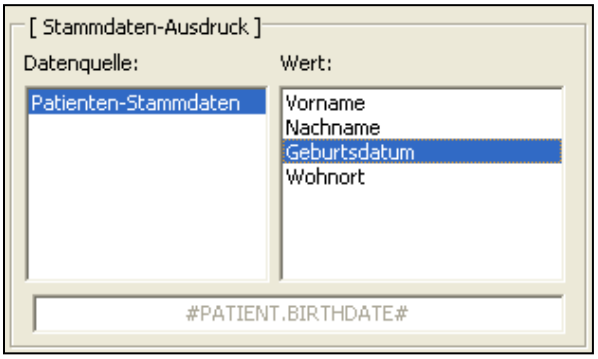


Abbildung 28: OTST – Screenshot der Stammdaten-Auswahl

Textgenerierung

Zu jedem Stammdatenausdruck wird eine SQL-Abfrage hinterlegt, die bei der Textgenerierung ausgeführt werden kann. Dazu ist in der Grammatik-Unit eine Konstante als zweistufiges String-Array mit Ausdruck und SQL-Abfrage definiert. Der Teilstring `#ID#` muss vor Ausführung der Abfrage nur noch durch den jeweiligen Identifier ersetzt werden. Welcher Wert eingesetzt wird, hängt vom Ausdruck selbst ab und kann nach dem Parsen des Ausdrucks bestimmt werden.

```
const
  Grammar_Master_SQL : array [0..2, 0..1] of string = (
    ('PATIENT.SURNAME',   'select U_PT_Surname as RETURNVAL from
                           TU_PatientPrivate where U_PT_ID=#ID#'),
    ('SURGEON.FIRSTNAME', 'select U_SG_Name as RETURNVAL from
                           TU_SG_Surgeon where U_SG_ID=#ID#')
  );
```

Für das Ersetzen eines Stammdaten-Ausdrucks durch den realen Wert kann der Ausdruck mit dem ersten Teil des Arrays `Grammar_Master_SQL` verglichen werden, dadurch erhält man das SQL-Statement im zweiten Teil. Der Generator muss anschließend nur noch den Teilstring `#ID#` durch den aktuell benötigten Wert ersetzen. Welcher Wert das ist, entscheidet der geparste Grammatik-Ausdruck. Die ID ist dabei immer ein Integer-Wert, der auf die Identity-Spalte einer Tabelle zeigt.

Der folgende Code-Abschnitt zeigt die Logik des Generators, die einen Stammdaten-Grammatik-Ausdruck auflöst. Der Einfachheit halber wird im Beispiel als ID immer die interne Patienten-ID verwendet, anstatt eine zusätzliche Unterscheidung aufgrund des geparsten Ausdrucks vorzunehmen.

```
var
  i      : Integer;
  sqlStr, resolved : String;
  qry    : TADOQuery;
begin
  // 1. Find SQL-Statement for the Expression
  i      := 0;
  sqlStr := '';
  while (Trim(sqlStr)='') and (i<High(Grammar_Master_SQL)) do
  begin
    if (Grammar_Master_SQL[i, 0] = expression) then
      sqlStr := Grammar_Master_SQL[i, 1]
    else
      Inc(i);
  end;

  // 2. Execute the SQL-Statement
  qry := TADOQuery.Create;
  // PT_ID is internal Patient-ID
  qry.SQL.Add(StringReplace(sqlStr,'#ID',IntToStr(PT_ID),[rfReplaceAll]));
  // catch potential error in SQL-Statement
  try
    qry.Open;
    resolved := qry.FieldByName('RETURNVAL').AsString;
  finally
    qry.Close;
  end;
  qry.Free;
end;
```

4.5.2. Berichtsdaten

Die Definition der Ausdrücke für Berichtsdaten besteht aus String-Konstanten, deren gültige Reihenfolge in der Parse-Methode des Generators definiert wird.

Die Textgenerierung für Berichtsdaten-Ausdrücke ist im Kapitel 4.7 beschrieben.

4.6. Generator

Die Logik des Brief-Generators ist die komplexeste des Moduls *Arztbriefgenerierung*. Die Generierung eines Briefes ist abhängig von der Vorlage, vom Patient, seiner Datengrundlage und einigen Programmkonstanten (die zum Verständnis der Logik nicht weiter wichtig sind).

Die Datengrundlage kennzeichnet eine Menge von Einträgen in der Krankengeschichte des Patienten, deren Werte in den Brief einfließen können. Der Aufruf der Generierung kann von verschiedenen Stellen aus erfolgen, beispielsweise kann ein Brief auf einem Operationsbericht basieren, auf einem Behandlungstag oder der gesamten Behandlungsgeschichte (identifiziert mit der CID, der Fall/Case-ID).

Der Generator wird in der Klasse `LtrGenerator` implementiert. Die Methode `startLetterCreation` legt die Datengrundlage aufgrund der Fall-ID fest. Als Parameter verlangt sie die Patienten-ID, die ID der Briefvorlage (`tmplID`), das Modul der Software und die Klinik-ID (`hpid`). Modul und Klinik-ID sind notwendig für die Zuordnung des erstellten Briefes innerhalb des Hauptprogramms OTST. Letzter Parameter ist die Fall-ID (engl. Case-ID, `cid`). Als Rückgabewert liefert die Funktion einen Verweis auf den erstellten Brief in der Krankengeschichte.

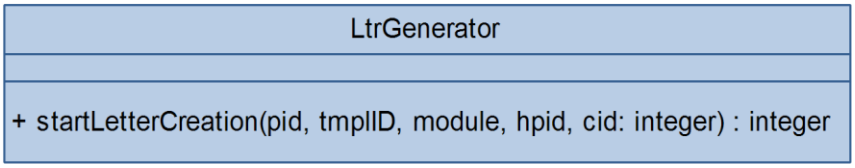


Abbildung 29: Generator mit öffentlichen Methoden

Die folgende Abbildung zeigt den internen Ablauf der Methode als Sequenzdiagramm. Nach Aufruf werden im Schritt 2 die Patientendaten geladen, im Schritt 3 die Datengrundlage. Die Datengrundlage wird hier durch die Fall-ID festgelegt. Durch die interne Speicherung der Einträge in einer Liste (`chList`, siehe Abbildung 31) ist es einfach möglich, weitere start-Methoden mit anderer Datengrundlage zu implementieren (z.B. nur ein Behandlungsgeschichten-Eintrag, Datumseingrenzung etc.). Schritt 4 lädt die Briefvorlage und im Schritt 5, dem Ersetzen der Grammatik-Ausdrücke durch reale Daten kommt die Textgenerierung ins Spiel, sie ist im Kapitel 4.7 beschrieben. Im Schritt 6 letztendlich wird der erstellte Brief in der Datenbank gespeichert und die Verweise zum Patienten erstellt.

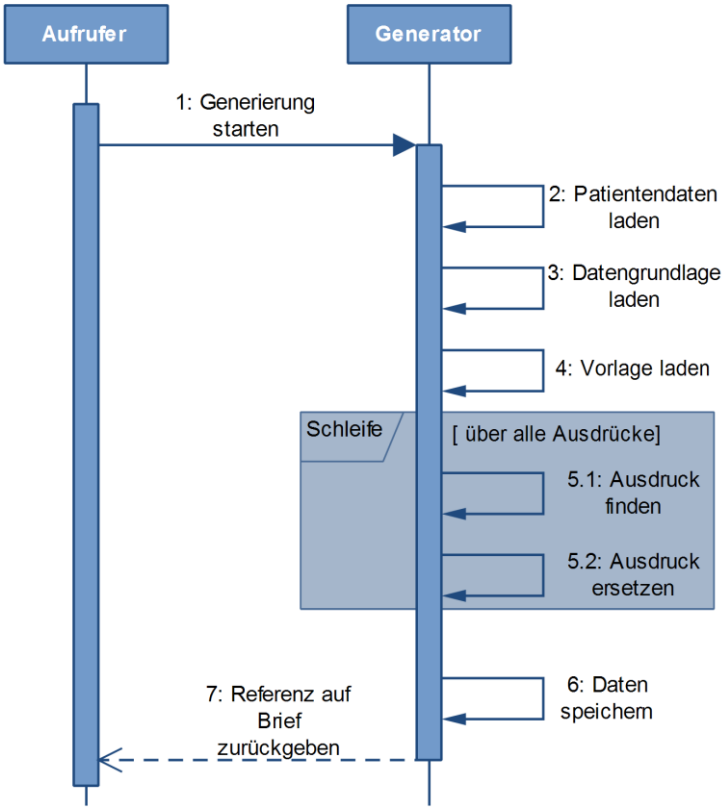


Abbildung 30: Generator, Sequenzdiagramm der Briefgenerierung

Abbildung 31 zeigt die privaten Methoden und Variablen, die der Generator während der Briefgenerierung verwendet. Nicht aufgeführt sind die Methoden zur Textgenerierung, dazu siehe Abbildung 32, Seite 63. Die Methode `resolveTemplate` (Schritt 5) ruft für jeden Grammatik-Ausdruck `parseExpr` und `resolveExpr` (Schritt 5.2, beschrieben im Kapitel 4.7) auf. Das Parsen des Ausdrucks hilft beim Ersetzen zur Unterscheidung, von welchem Typ der Ausdruck ist (Stammdaten, Berichtsdaten). Dazu wird ein weiterer hier nicht näher erklärter Objekttyp definiert, der den gesamten Ausdruck und mehrere Variablen für die Unterscheidung enthält. Die Methode `init_Patient_data` lädt die Patientendaten, `init_Template_data` die Vorlagendaten. `Init_CHCID_data` lädt die Datengrundlage aufgrund einer Fall-ID.

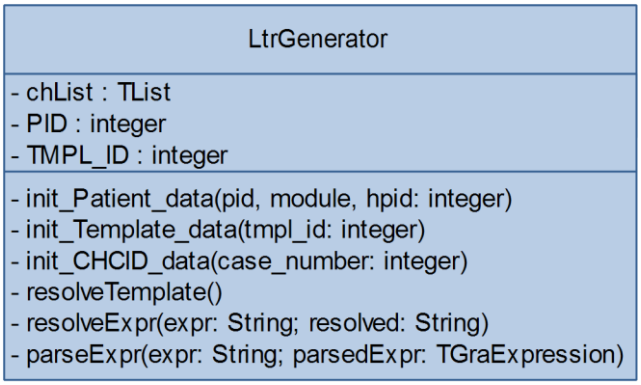


Abbildung 31: Generator mit einem Teil der privaten Variablen und Methoden

4.7. Textgenerierung

Die Textgenerierung (im Sequenzdiagramm in Abbildung 30 Schritt 5.2) ist direkt abhängig von der Grammatik und der bestehenden Datenbankstruktur. Sie kann auf zwei Bereiche aufgeteilt werden:

- Stammdaten, im Kapitel 4.5.1 beschrieben.
- Berichtsdaten (MCS-Ausdruck)

4.7.1. Ablauf

Die Generierung von Text für die flexiblen Berichtsdaten ist ungleich komplizierter als die der Stammdaten. Hier kann ein Ausdruck, abhängig von der Anzahl Kind-Ebenen und der letztendlich aktivierten Elemente aus einem oder mehreren Textbausteinen bestehen. Die Definition der MCS-Codes (vorgegeben vom Hauptprogramm) ist außerdem so, dass aus einem MCS-Code nicht hervorgeht, welche Art Bericht (Operation, Bild, PDF) er referenziert. Die folgende Liste zeigt das Prinzip der Auflösung eines MCS-Ausdrucks:

1. Relevante Berichte in eine Liste laden (`chList`, siehe Abbildung 31)
2. Aktivierte Einträge aller relevanten Berichte in einen XML-Baum laden
3. XML-Baum nach dem Knoten mit gesuchtem MCS-Code durchsuchen
4. Werte des Knotens abhängig vom MCS-Ausdruck auslesen und zusammensetzen

Schritt 1 wird vom Generator schon zu einem früheren Zeitpunkt ausgeführt (Abbildung 30 Schritt 3, hier werden Typ (Operation, Bild, PDF etc.) und Datenbank-ID jedes Berichts in eine Liste geschrieben). Im **Schritt 2** wird ein XML-Baum (DOM) erstellt, in den die aktivierten Einträge alle Berichte mit hinterlegter Struktur einfließen. Eine Baumstruktur bietet den Vorteil, dass sie für das Zusammensetzen von Text einfach durchlaufen werden kann, auch im Hinblick auf die optionalen Kind-Ebenen. Der XML-DOM-Baum von `TNativeXML` (`NativeXML.TXMLNode` als Knotentyp) bietet hier alle Funktionalitäten, die man benötigt, ohne eine eigene Baumstruktur implementieren zu müssen.

Schritt 3: Ist der Baum einmal erstellt, wird mit `NativeXML.Root.NodeByAttributeValue` der gesuchte Knoten inklusive seiner Kind-Knoten mit einem Aufruf gefunden (gesucht wird nach dem MCS-Code, der im Attribut `code` liegt, siehe folgenden Code-Abschnitt). Im **Schritt 4** wird, abhängig vom geparsten MCS-Ausdruck (Format, Content, Kind-Ebenen) ein String erstellt, der alle gewünschten Werte enthält.

Der folgende Code-Abschnitt zeigt die Struktur des erwähnten XML-Baums. Wurzelelement ist `MCSdata`, darunter folgen beliebig verschachtelt die `item`-Elemente. Der Einfachheit halber wird hier nur das Attribut `code` aufgeführt. In diesem Beispiel enthält der XML-Baum zwei Behandlungseinträge: AUOP (für Operation) und IMG (für Bild). Für die Operation ist der Eintrag THER aktiviert (Therapie), darunter OSSI (Ossikuloplastik) und STAP (Stapedotomie), weiters eine Diagnose. Für das Bild ist ein Messwert aktiviert.

```
<MCSdata>
  <item code="AUOP" date="2011-01-01">
    <item code="THER">
      <item code="OSSI">
        <item code="STAP"/>
      </item>
    </item>

    <item code="DIAG">
      <item code="OMCS"/>
    </item>
  </item>

  <item code="IMG" date="2011-01-02">
    <item code="MEASI">
      <item code="CANX"/>
```

```
    </item>
  </item>
</MCSdata>
```

Jedes Element `item` enthält in Attributen alle Werte, die möglicherweise für das Auflösen eines MCS-Ausdrucks benötigt werden. Je nach Content-Ausdruck kann dies ein anderer Wert sein: Name, Textbaustein, Numerischer oder Textinhalt. Das Attribut `datatype` definiert den Datentyp des Eintrags (Einfacher Eintrag also reine Auswahl, Textwert, Numerischer Wert). Das Attribut `viewtype` ist eine Definition der Berichtstruktur des Hauptprogramms, die zusätzlich bei der Textgenerierung verwendet wird. Die Attribute `val_num` und `val_text` enthalten den numerischen Wert bzw. den Textinhalt, `name` den Namen des Eintrags und `descr` den hinterlegten Textbaustein.

```
<item code="OMCS" name="OMC simplex" descr="OMC vom Typ simplex"
      val_num="" val_text=""
      datatype="2" viewtype="1" />
```

4.7.2. Methoden

Die beschriebenen Schritte werden in privaten Methoden der Generatorklasse durchgeführt. Abbildung 32 zeigt die Methoden mit Parametern. Eine detaillierte Erklärung jeder Methode mit Parametern ist an dieser Stelle nicht sinnvoll, das Prinzip wird jedoch als Überblick erklärt.

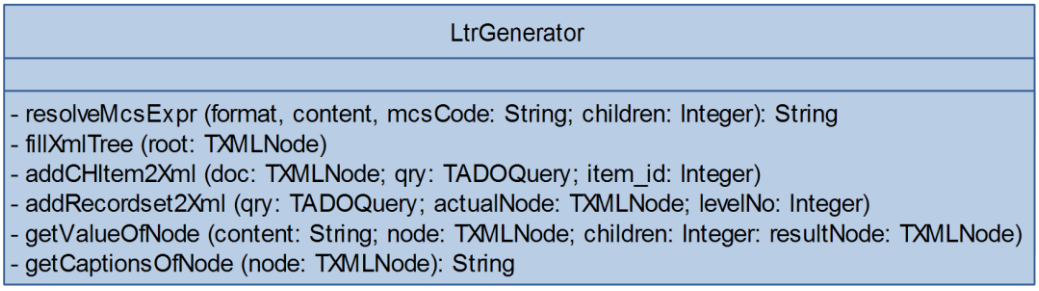


Abbildung 32: Generator mit privaten Methoden für die Textgenerierung

Überblick

Die beschriebenen Schritte werden gestartet in der Methode `resolveMcsExpr`. Diese wird aufgerufen von der Methode `resolveExpr` (siehe Abbildung 31). Das Erstellen des XML-Baums geschieht in der Methode `fillXmlTree`, diese ruft für jeden Eintrag der Krankengeschichte (Datengrundlage) die Methode `addCHItem2Xml` auf (add Clinical-History-Item to XML). Diese Methode führt die Transformation „Tabellen einer relationalen Datenbank nach XML“ durch. Über eine SQL-Abfrage erhält sie eine Datensatzmenge, die von der Methode `addRecordset2Xml` rekursiv abgearbeitet wird.

Nachdem der Knoten im erstellten XML-Baum aufgrund des MCS-Codes gefunden wurde, wird er von der Methode `getValueOfNode` im Schritt 4 rekursiv durchlaufen und abhängig von Content und Kind-Ebenen wird ein neuer XML-Baum (Call-by-Reference Parameter `resultNode`) erstellt. Dieser wiederum wird von der Methode `getCaptionsOfNode` durchlaufen und der endgültige String erstellt.

Der Ergebnis-Baum der Methode `getValueOfNode` enthält beliebig verschachtelte `item`-Elemente mit folgenden Attributen:

```
<item caption="OMC Typ simplex" div1=", " div2="." viewtype="1">
  <item .../>
</item>
```

Das Attribut `caption` enthält den gewünschten Wert, der die Methode `getValueOfNode` aufgrund des `content` Parameters erstellt hat. Aufgrund des Datentyps (`datatype`) werden die Verbindungsstrings (`div1`, `div2`) erstellt. Die Methode `getCaptionsOfNode` fügt nun alle

`captions`-Werte zusammen, verbunden durch die Verbindungsstrings, abhängig vom `viewtype` kommen noch weitere Zeichen hinzu.

Formatierung

Die Methode `resolveMcsExpr` setzt nun den Wert des Attributs `caption` des Resultatknotens und dessen Kindknoten aufgrund des Formats zusammen. Zum Beispiel bedeutet die Formatangabe „DESCR“ Fließtext, also werden die Werte durch Punkt getrennt zusammengesetzt. „LISTSORT“ bedeutet eine sortierte Liste, also beginnt der Resultatstring hier mit dem Wert `<list listType="unsorted">`, und die Werte der Kindknoten werden als Listenelement `<item> … </item>` hinzugefügt. Am Ende wird die HL7-CDA-Liste mit `</list>` geschlossen und der Resultatstring kann von der aufrufenden Methode direkt in den Brief eingefügt werden.

5. Zusammenfassung / Beurteilung

Mit der vorliegenden Arbeit wurde für die bestehende Software *Otis – ENTstatistics* ein Modul zur *Arztbriefgenerierung* entworfen und implementiert. Im ersten Teil dieses Kapitel werden die Erkenntnisse aus den Grundlagen zusammengefasst, im zweiten Teil die entworfene Architektur mit der Aufteilung in Komponenten. Am Ende wird die erreichte Entwicklung betrachtet und im Ausblick die noch fehlenden Schritte skizziert, die für das Release des Moduls notwendig sind.

5.1. Erkenntnisse aus den Grundlagen

Die Erkenntnisse aus den Grundlagen lassen sich zusammenfassen in die Bereiche Arztbriefschreibung und die nationalen Standardisierungsbemühungen. Der dritte Teil beschreibt den Ansatz, um diese Theorien in einem Produkt aus der Praxis umzusetzen.

5.1.1. Arztbriefschreibung

Das Verfassen eines Arztbriefes mit dem Zweck der Information einer weiterbehandelnden Organisation ist in einem Rahmen zu sehen, der die gesamte Behandlung eines Patienten umfasst. Sie beginnt beim ersten Gang des Patienten zum Hausarzt und schließt Untersuchungen und Therapien aller beteiligten Organisationen mit ein. Der Arztbrief zum Zeitpunkt der Entlassung des Patienten aus dem Krankenhaus stellt eine Zusammenfassung der Behandlungsgeschichte mit Handlungsempfehlungen dar.

Aufgrund der umfassenden Dokumentationspflichten in Krankenhäusern ist ein Großteil der Informationen, die später in den Arztbrief einfließen, zum Zeitpunkt der Entlassung des Patienten bereits in Softwaresystemen erfasst. Es ist die Aufgabe von Softwareherstellern, den Arzt mit geeigneten Softwaretools beim Verfassen des Arztbriefes dahingehend zu unterstützen, dass die benötigten Informationen automatisch zusammengefasst werden und er, im Idealfall, nur noch Schönheitskorrekturen am Text oder persönliche Empfehlungen anbringen muss.

Dadurch ergibt sich eine Zeitersparnis, die dem Arzt mehr Zeit für seine eigentliche Arbeit, der medizinischen Betreuung von Patienten gibt. Außerdem gelangt der Arztbrief so schneller zum Empfänger und die Weiterbehandlung des Patienten wird nicht verzögert.

5.1.2. Standardisierung

Die Weitergabe von Dokumenten an andere Organisationen ist in einem größeren Kontext zu sehen. Nationale Strategien wie *eHealth Schweiz* beschäftigen sich mit dem Austausch der gesamten Patientenakte. Unabhängig vom Ort der Behandlung soll auf alle Informationen zugegriffen werden können, die den Patienten und seine Krankheit betreffen. Dazu muss eine Architektur geschaffen werden, die Austauschmodi, Zugangskontrolle, Identifikation und Verantwortung definiert. Mit der erwähnten Strategie wurden die ersten Schritte in die Wege geleitet, damit alle Beteiligten im Gesundheitswesen (Krankenhäuser, Krankenkassen, niedergelassene Ärzte, Softwarehersteller) dazu beitragen, dieses Ziel zu erreichen.

Eine zwingende Voraussetzung für das Funktionieren eines zukünftig automatischen Datenaustausches ist die Standardisierung von Dokumenten. Dazu hat *HL7 International* ein Dokumentenformat im XML-Format definiert, das medizinische Dokumente standardisiert. *HL7 Schweiz* hat das Format an die nationalen Gegebenheiten angepasst, und definiert den HL7-CDA-CH-II Standard als empfohlenes Format für Briefe im Gesundheitswesen.

5.1.3. Produkt

Die Standardisierung bleibt so lange ein rein theoretisches Konstrukt, bis sie in ein Softwaresystem eingebunden wird, das in der Praxis verwendet wird. Die Software *Otis – ENTstatistics* wird an mehreren HNO-Kliniken in der Schweiz seit mehreren Jahren verwendet. Die Struktur der Datenbank und das Kernkonzept der Software berücksichtigen keine Standards aus dem Gesundheitswesen, die grundlegende Idee der Software ist die Flexibilität ihrer Berichtstruktur. Sie ermöglicht es den Kliniken jedoch, die gesamte Krankengeschichte eines Patienten in einem System abzulegen.

5.2. Systemarchitektur

Im Rahmen dieser Arbeit wurde ein Softwaremodul für *Otis – ENTstatistics* entwickelt, das die Arztbriefschreibung weitgehend automatisiert und außerdem Schnittstellen bietet, um Dokumente mit anderen Programmen in standardisierter Form auszutauschen.

Das Modul wird aufgeteilt in die drei Komponenten Viewer, Generator und Transformator. Sie werden in den Formularen, die rund um die Use Cases entwickelt werden, eingebunden und sind so entworfen, dass sie wiederverwendet und gegebenenfalls ausgetauscht werden können.

Abbildung 33 zeigt eine Übersicht über Use Cases, Komponenten und Daten. Auf der linken Seite sind die Use Cases dargestellt, die in den Aufgabenbereich des Systemadministrators fallen und auf der rechten Seite der Aufgabenbereich des Arztes. In der Mitte der Abbildung sind die Komponenten mit ihrer Haupttechnologie und die Daten, auf die sie zugreifen, dargestellt.

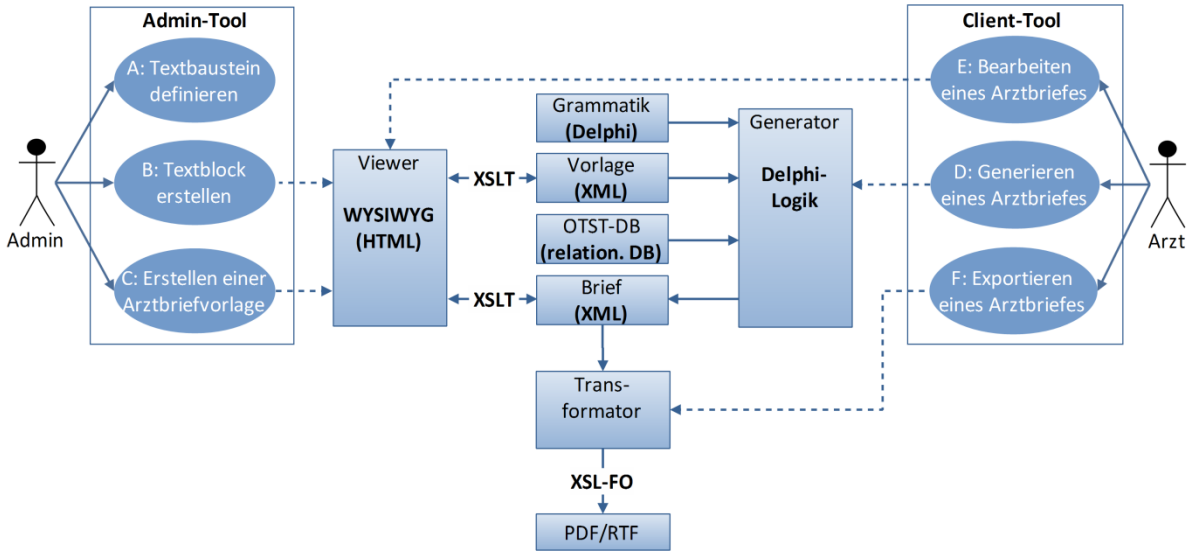


Abbildung 33: Modul *Arztbriefgenerierung* - Use Cases, Komponenten, Technologien

Die Viewer-Komponente wird in drei verschiedenen Use Cases (für die Administration und die Bearbeitung des Briefes) verwendet. Sie stellt sowohl die Briefvorlage, Textblöcke (Teil der Vorlage) als auch den Brief selbst dar. Sie ist als WYSIWYG-HTML-Editor entworfen und transformiert die XML-Eingangsdaten mit XSL-Stylesheets in HTML und wieder zurück.

Die Grammatik definiert Platzhalter, die in den Vorlagen enthalten sind und bei der Generierung durch realen Text ersetzt werden. Dies ist die Aufgabe des Generators. Dazu liest er die Krankengeschichte aus der relationalen OTST-Datenbank ein und generiert aus den strukturiert vorliegenden Daten Text. Am Ende erstellt er ein valides XML-Dokument nach dem HL7-CDA-CH-II Standard. Die Generierung von Text aus der Datenbank ist einer der komplexesten Teile des Softwaremoduls.

Um die erstellten Dokumente mit den bei Kunden aktuell installierten Softwaresystemen auszutauschen, können sie vom Transformator in die Formate PDF und RTF exportiert werden. Dazu bietet sich das XSL-FO Prinzip an, was prinzipiell nur bedeutet, dass ein Eingangs-XML-Dokument mit Hilfe von Stylesheets und einem XSL-FO-Prozessor in ein anderes Format transformiert wird.

5.3. Beurteilung des Softwaremoduls

Von der entworfenen Systemarchitektur wurden bis auf die Transformation in PDF/RTF alle Komponenten vollständig implementiert. Die folgende Aufzählung betrachtet die erreichte Entwicklung.

- Durch den Generator verfügt die Hauptsoftware nun über eine Funktionalität, um aus allen im System abgelegten Daten Text zu generieren. Die Generator-Komponente liest strukturiert hinterlegte Informationen ein und erstellt daraus Fließtext. Die Logik des Generators ist flexibel aufgebaut, so dass Zusammenfassungen entweder für einzelne Berichte oder auch die ganze Krankengeschichte mit denselben Funktionen erstellt werden können.
- Die Definition des zu generierenden Textes geschieht in einer eigens definierten Grammatik, die zwischen Stamm- und flexiblen Berichtsdaten unterscheidet. Sie wurde so entworfen, dass sie durch weitere Elemente erweitert werden kann, falls das Hauptprogramm erweitert werden sollte.
- In Briefvorlagen kann definiert werden, welche Informationen aus der Krankengeschichte des Patienten in den Brief einfließen sollen. Der Arzt als Anwender der Software kann also einmal verschiedene Vorlagen für verschiedene Empfänger (Hausarzt, anderes Krankenhaus, Patient selbst etc.) definieren und erhält per Mausklick alle benötigten Daten in Briefform.
- Der Viewer stellt den Brief nach dem WYSIWYG-Prinzip dar, d.h. der Anwender sieht Vorlage und Brief so formatiert, wie er beim Ausdruck erscheinen wird.
- Durch Implementierung eines Viewer-Fensters, das ein XML-Dokument im CDA-Standard oder dazu ähnlich (konkret muss nur das Body-Element `component` standardkonform sein) als Eingangsdokument akzeptiert, konnte diese Komponente einfach in drei verschiedene Formulare (Editor für Textblöcke, Briefvorlagen, Briefe) eingebaut werden. Mit Hilfe der implementierten CDA-Klasse kann das vom Viewer erstellte Ausgangsdokument einfach durch jene Elemente ergänzt werden, damit ein CDA-konformes XML-Dokument entsteht.
- Durch die interne Speicherung des Briefes im HL7-CDA-Format unterstützt die Hauptsoftware nun einen internationalen Standard des Gesundheitswesens. Damit ist eine weitere Schnittstelle geschaffen, was die Anbindung an andere Programme vereinfacht und die Software für neue Kunden interessanter macht.
- Die Speicherung von XML-Dokumenten als reine Strings in der Datenbank scheint eine primitive Lösung zu sein. Sie ist jedoch zweckmäßig, da im aktuellen Umfeld keine erweiterten Anforderungen beim Lesen und Schreiben der XML-Daten gestellt werden. Die XML-Strings lassen sich außerdem einfach in die XML-Datentypen von Delphi transformieren. Durch zentrale Speicherung aller XML-Daten in einer Tabelle und zentrale Schreib- und Lesemethoden im Programm hat man außerdem die Möglichkeit, dieses Prinzip mit wenig Aufwand anzupassen, z.B. unter Verwendung des XML-Datentyps von MS SQL Server.
- Durch die Aufteilung des Moduls in Komponenten mit klar definierten Schnittstellen und Aufgabenbereichen wurde eine Anwendung entwickelt, die sich einfach warten und erweitern lässt. Jedes der Module (Viewer, Generator, Transformator) könnte ausgetauscht oder in anderen Anwendungen wieder verwendet werden.
- Das Modul hat bisher noch keine formalen Tests durchlaufen. Getestet wurden die funktionalen Klassen durch GUI-Tests der realisierten Benutzeroberfläche (anhand der beschriebenen Use Cases). Die Korrektheit der generierten XML-Dokumente und des generierten Textes wurde anhand der üblicherweise in der Firma verwendeten Testdatenbank überprüft, die auch für Demonstrationszwecke für Kunden verwendet wird. Die in Zwischenschritten erstellten XML-Bäume wurden zu Testzwecken in Dateien geschrieben und überprüft.

5.4. Ausblick

Vor dem Release des Moduls *Arztbriefgenerierung* sind die folgenden Schritte notwendig:

Implementierung des Transformator

Um die Transformator-Logik zu implementieren, können die in den Grundlagen aufgezeigten XSL-FO-Prozessoren verwendet werden. Die Formatangaben müssen dazu mit der Vorlage hinterlegt und bei der Generierung des Briefes verlinkt werden. Für die Durchführung der Transformation benötigt es danach noch ein XSL-Stylesheet, um das CDA-Dokument in ein XSL-FO-Dokument zu transformieren.

Alternativ kann die Delphi-Komponente von FastReports (siehe Kapitel 2.4.1) verwendet werden. Der Inhalt des Briefes wird an definierten Positionen einer Seite eingefügt und für die Transformation in PDF und RTF kann die Funktionalität der Komponente verwendet werden. Oder es wird eine andere Delphi-Komponente gesucht, die in der Lage ist, HTML in die beiden Formate umzuwandeln.

Funktionale Tests

Vor dem Release müssen Tests durchgeführt und dokumentiert werden. In der Firma werden Tests üblicherweise nur GUI-basiert durchgeführt. Das heißt, es werden Anwendungsfälle entworfen, und diese mit Hilfe von Datenbanken aus der Praxis durchlaufen. Key-Kunden stellen dafür ihre Datenbanken mit anonymisierten Daten zur Verfügung.

Einbindung in das Hauptprogramm

Anschließend kann das Modul *Arztbriefgenerierung* in den Funktionsumfang von OTST aufgenommen werden. Dazu muss ein SQL-Skript erstellt werden, das die zusätzlichen Tabellen zur vorhandenen Datenbank hinzufügt. Die in einem eigenen Branch (SVN-Repository) implementierten Delphi-Klassen müssen in den Hauptzweig der Entwicklung übernommen und als standardmäßig deaktiviertes Modul eingebunden werden.

Dokumentation

Für die Dokumentation sind die folgenden Aufgaben durchzuführen: Die Funktionalität aus Anwendersicht muss im Benutzerhandbuch mit Anwendungsbeispielen beschrieben werden. Für die technische Dokumentation müssen die Tabellen und ihre Spalten in MS SQL-Server beschrieben werden, daraus wird eine HTML-Dokumentation generiert. Die Beschreibung der Architektur kann komplett aus dieser Arbeit übernommen werden. Aus dem bereits dokumentierten Quellcode kann eine HTML-Dokumentation der Klassen generiert werden.

Referenzen

Adunka, O. (2004). *AdOnco - Ein klinisch-wissenschaftliches Datenbanksystem zur Erfassung und Auswertung onkologischer Daten im Kopf-Hals-Bereich.* Tübingen: Springer Verlag.

Altsoft. (31.12. 2010). *Altsoft*. Abgerufen am 09.01.2011 von Altsoft Onlineshop: http://www.alt-soft.com/Shop.aspx

Apache-Software-Foundation. (11.09.2010). *Apache Software Foundation*. Abgerufen am 07.01.2011 von Apache FOP: http://xmlgraphics.apache.org/fop/index.html

BAG. (2007). *Strategie "eHealth" Schweiz.* Bern: Bundesamt für Gesundheit.

Bsalsa. (31.12. 2009). *Bsalsa*. Abgerufen am 15.11.2010 von Bsalsa Embedded WebBrowser: http://www.bsalsa.com/

Computerführer. (09.01.2011). *Computerführer für Ärzte*. Abgerufen am 09.01.2011 von Computerführer für Ärzte: http://www.computerfuehrer.de/modules/AMS/article.php?storyid=120

DIMDI. (11.01.2011). *DIMDI*. Abgerufen am 11.01.2011 von DIMDI ICD10: http://www.dimdi.de/static/de/klassi/diagnosen/icd10/

DIMDI. (11.01.2011). *DIMDI*. Abgerufen am 11.01.2011 von DIMDI - OPS: http://www.dimdi.de/static/de/klassi/prozeduren/ops301/index.htm

eHealth. (2008). *Aufträge Teilprojekte Strategie "eHealth".* Bern: Koordinationsorgan eHealth.

ehealthsuisse. (2009). *Standards und Architektur - Erste Empfehlungen.* Bern: Koordinationsorgan ehealthsuisse.

FSF. (27.06.2007). *GNU*. Abgerufen am 09.01.2011 von GNU Lesser General Public License: http://www.gnu.org/licenses/lgpl.html

Gelbe-Seiten. (14.12.2010). *Gelbe Seiten*. Abgerufen am 14.12.2010 von Gelbe Seiten: http://yellow.local.ch/de/q/St.%20Gallen/Arzt.html

Hauser, T. (2006). *XML Standards schnell + kompakt.* Frankfurt: entwickler.press.

HL7. (15.12.2010). *HL7*. Abgerufen am 15.12.2010 von HL7 FAQs: http://www.hl7.org/about/FAQs/index.cfm?ref=footer

HL7-Schweiz. (09.03.2009). *HL7 Benutzergruppe Schweiz.* Abgerufen am 01.12.2010 von CDA-CH-II: Spezifikation zum Erstellen von HL7 CDA Templates: http://www.hl7.ch/fileadmin/ungeschuetzte_dateien/files_xepr/CDA-CH-II_de_V1.1a.pdf

HL7-Schweiz. (16.12.2010). *HL7 Benutzergruppe Schweiz*. Abgerufen am 16.12.2010 von HL7 Benutzergruppe Schweiz - HL7: http://www.hl7.ch/hl7-benutzergruppe-schweiz.html

IHE. (16.12.2010). *IHE*. Abgerufen am 16.12.2010 von About IHE: http://www.ihe.net/About/index.cfm

ihe-suisse. (14.12.2010). *ihe-suisse*. Abgerufen am 14.12.2010 von ihe-suisse: http://www.ihe-suisse.ch

ihe-suisse. (2010). *Statuten.* St. Gallen: IHE Suisse.

innoForce. (11.01.2011). *innoForce*. Abgerufen am 11.01.2011 von innoForce: http://www.innoforce.com/de

JasperForge. (12.31.2010). *JasperForge*. Abgerufen am 09.01.2011 von JasperReports: http://jasperforge.org/index.php?q=project/jasperreports

Kaluschka, R. (2005). *Informationsgewinnung aus Freitexten in der Rehabilitationsmedizin.* Köln: Diplomarbeit.

Kantonsrat-Zürich. (2004). Patientinnen- und Patientengesetz. Schweiz: Kantonsrat Zürich.

KBV. (11.01.2011). *KBV*. Abgerufen am 11.01.2011 von KBV - IT in der Praxis - Schnittstellen: http://www.kbv.de/ita/4274.html

Koch, D. (2007). *XSLT schnell + kompakt.* Frankfurt: entwickler.press.

krankenhaus.ch. (14.12.2010). *krankenhaus.ch*. Abgerufen am 14.12.2010 von krankenhaus.ch: http://www.krankenhaus.ch/krankenhaus/index.php

Krüger, M., & Welsch, U. (2007). *XSL-FO Praxis schnell + kompakt.* Frankfurt: entwickler.press.

NEMA. (11.01.2011). *DICOM Homepage.* Abgerufen am 11.01.2011 von DICOM Homepage: http://medical.nema.org/dicom/geninfo/Strategy.pdf

Oestereich, B. (2009). *Analyse und Design mit UML 2.3.* München: Oldenbourg Wissenschaftsverlag GmbH.

Preece, J. (1994). *Human-Computing Interaction.* München: Addison-Wesley.

ProfGrid. (31.12. 2010). *ProfGrid*. Abgerufen am 15.11.2010 von ProfGrid.com Order form: https://secure.plimus.com/jsp/branded_store.jsp?developerId=36229

ProfGrid. (31.12.2010). *ProfGrid*. Abgerufen am 15.11.2010 von ProfDHTMLEdit: http://www.profgrid.com/dhtmledit.html

RenderX. (01.07.2010). *XSL-FO.* Abgerufen am 07.01.2011 von Published Price List: http://www.renderx.com/files/documents/RenderX_Published_Price_List.pdf

SAP. (08.01.2011). *SAP Business Objects*. Abgerufen am 08.01.2011 von SAP Europe: http://store.businessobjects.com/store/bobjemea/DisplayHomePage/Locale.de_DE/Currency.EUR

Semler, S. C. (2001). Automatisierte Arztbriefschreibung - Wie weit darf sie gehen? *Der Computerführer für Ärzte* , S. 102.

Stichnote, M. (2006). *Anforderungen von Solvency II an das operationelle Risikomanagement in Versicherungen - Konzeption und Ausgestaltung eines Risikoberichtssystems.* Norderstedt: Diplomarbeit.

swissdrg. (01.02.2010). *swissdrg*. Abgerufen am 10.12.2010 von swissdrg: https://webgrouper.swissdrg.org/

theunknownones. (15.11.2010). *theunknownones*. Abgerufen am 15.11.2010 von theunknownones: http://code.google.com/p/theunknownones/

tieto. (11.01.2011). *iMedOne*. Abgerufen am 11. 01 2011 von iMedOne: http://tieto.de/branchen/healthcare/KIS/medizinische-workflows/imedone-medizinische-dokumentation

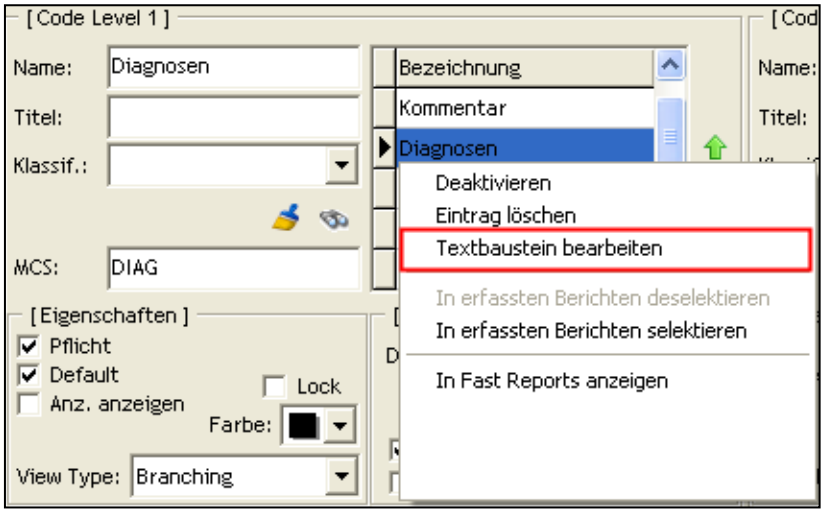
VHitG. (12.05.2006). *Verband der Hersteller von IT-Lösungen für das Gesundheitswesen e.V.* Abgerufen am 16. 12 2010 von Implementierungsleitfaden Arztbrief: http://download.vhitg.de/Leitfaden-VHitG-Arztbrief-v150.pdf

W3C. (06.12.2006). *World Wide Web Consortium*. Abgerufen am 07.01.2011 von XSL Transformations: http://www.w3.org/TR/xslt

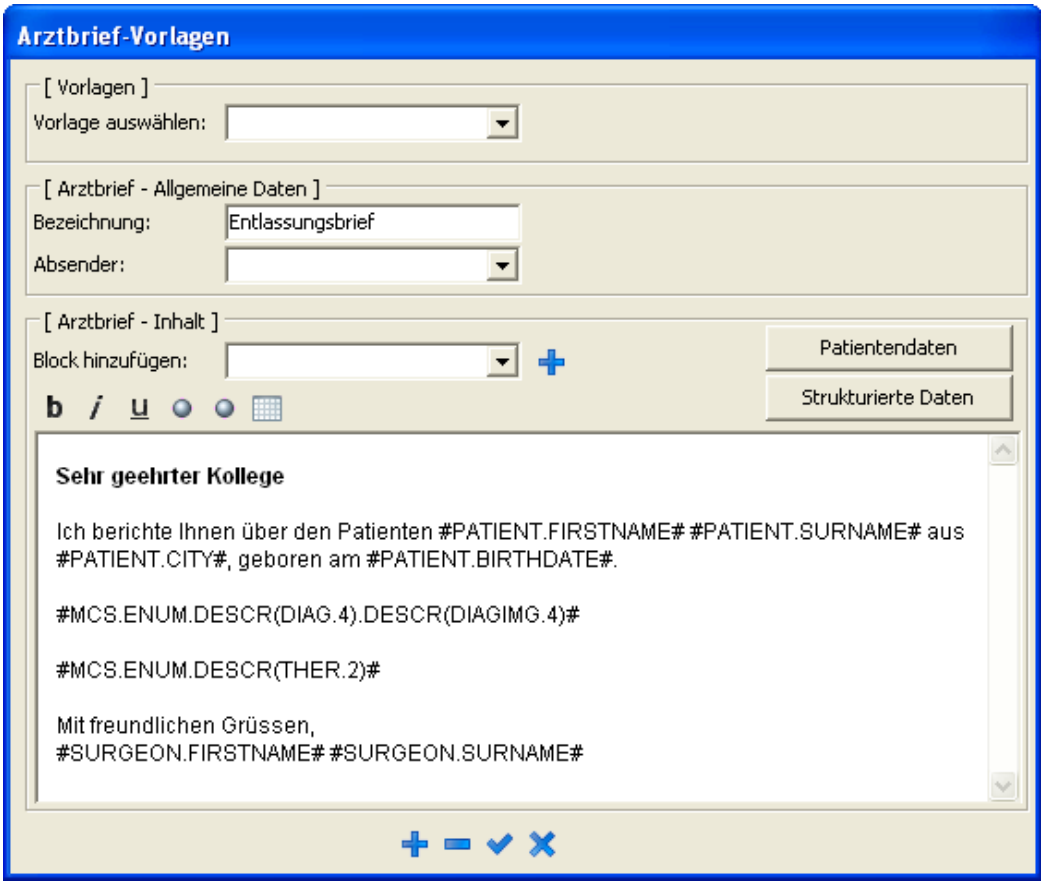
W3C. (20.10.2009). *World Wide Web Consortium*. Abgerufen am 07.01.2011 von XML Core Working Group: http://www.w3.org/XML/Core/#Publications

Anhang 1: Screenshots

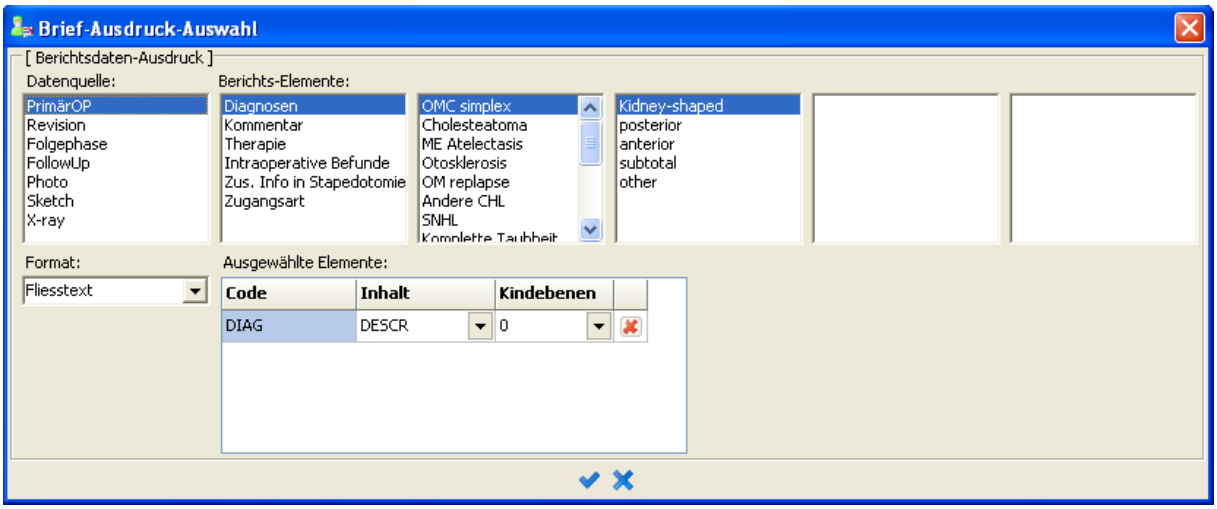
Die folgende Abbildung zeigt ein Teilfenster der Berichtsstruktur im Admin-Tool zur Definition eines Textbausteins (Use Case A).



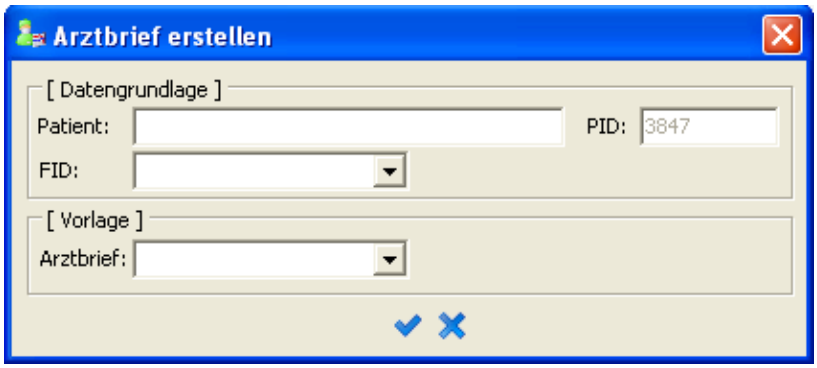
Die folgende Abbildung zeigt das Fenster zur Bearbeitung von Arztbriefvorlagen (verkleinerter Texteditor) mit beispielhaftem Inhalt. Im oberen Teil die Dropdown-Liste zur Auswahl einer vorhandenen Vorlage, im unteren Teil die Buttons zum Neuerstellen, Löschen, Speichern und Abbrechen. Auf der rechten Seite zwei Buttons zum Einfügen von Grammatik-Ausdrücken.



Die folgende Abbildung zeigt das Fenster zur Auswahl eines Grammatik-Ausdrucks (Berichtsdatenausdruck, das Fenster für Stammdaten zeigt die Abbildung 28). Oben links die Liste mit den verfügbaren Berichten, rechts davon für den ausgewählten Bericht dessen Einträge in bis zu 5 Unterebenen. Im unteren Teil werden die mit Doppelklick ausgewählten Einträge mit Code und wählbarem Inhalt aufgeführt, links davon die gewünschte Formatierung.



Die folgende Abbildung zeigt das Fenster mit den Auswahlmöglichkeiten bei Generierung eines Arztbriefes. Der aktivierte Patient wird im obersten Textfeld mit Namen und PID angezeigt, der Benutzer muss FID und Vorlage auswählen und startet mit dem Häkchen-Icon die Generierung.



Anhang 2: XML

Der folgende Code-Abschnitt zeigt einen beispielhaften Arztbrief im HL7-CDA-CH-II Standard:

```
<?xml version="1.0" encoding="UTF-8"?>

<ClinicalDocument
    xmlns="urn:hl7-org:v3" xmlns:voc="urn:hl7-org:v3/voc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:schemaLocation="urn:hl7-org:v3 /processable/coreschemas/CDA.xsd">

  <typeId root="2.16.840.1.113883.1.3" extension="POCD_HD000040"/>
  <id root="1.2.3.4.5.12345678.1" extension="12"/>
  <code code="11505-5" codeSystem="2.16.840.1.113883.6.1"
        displayName="Test-Dokument"/>

  <title>Test-Dokument</title>
  <effectiveTime value="20110101150000"/>
  <confidentialityCode code="N" codeSystem="2.16.840.1.113883.5.25"/>
  <languageCode code="de-CH"/>

  <recordTarget>
    <patientRole>
      <id extension="123" root="1.2.3.4.5.12345678"/>
      <addr>
        <streetName>Strasse</streetName>
        <houseNumber>12</houseNumber>
        <postalCode>9496</postalCode>
        <city>Balzers</city>
      </addr>
      <telecom nullFlavor="NASK"/>
      <patient>
        <name>
          <given>Hans</given>
          <family>Muster</family>
        </name>
        <administrativeGenderCode code="M"
                                  codeSystem="2.16.840.1.113883.5.1"/>
        <birthTime value="19800101"/>
      </patient>
    </patientRole>
  </recordTarget>

  <author>
    <functionCode code="PRISURG" codeSystem="2.16.756.5.30.2.1.1.1"/>
    <time value="20100928000000"/>
    <assignedAuthor>
      <id extension="10" root="1.2.3.4.5.12345678"/>
      <assignedPerson>
        <name>
          <prefix qualifier="AC">Dr.med.</prefix>
          <given>Peter</given>
          <family>Muster-Doc</family>
        </name>
      </assignedPerson>
    </assignedAuthor>
  </author>

  <custodian>
    <assignedCustodian>
      <representedCustodian>
        <id extension="11" root="1.2.3.4.5.12345678"/>
        <telecom nullFlavor="NASK"/>
```

```
        <addr>
          <streetName>Neugruet</streetName>
          <houseNumber>9</houseNumber>
          <postalCode>9496</postalCode>
          <city>Balzers</city>
        </addr>
      </representedCustodian>
    </assignedCustodian>
  </custodian>

  <informationRecipient>
    <intendentRecipient>
      <informationRecipient>
        <name>
          <prefix>Dr.med.</prefix>
          <given>Sandra</given>
          <family>Muster-Doc</family>
        </name>
      </informationRecipient>
      <receivedOrganization>
        <addr>
          <streetName>Heilgkreuz</streetName>
          <houseNumber>18</houseNumber>
          <postalCode>9490</postalCode>
          <city>Vaduz</city>
        </addr>
      </receivedOrganization>
    </intendentRecipient>
  </informationRecipient>

  <component>
    <structuredBody>
      <component>
        <section>
          <title>Diagnose</title>
          <text>OMC simplex, subtotal usw.</text>
        </section>
      </component>

      <component>
        <section>
          <title>Therapie</title>
          <text>Primaer OP usw.</text>
        </section>
      </component>
    </structuredBody>
  </component>
</ClinicalDocument>
```

Eidesstattliche Erklärung

Ich versichere, dass die vorliegende Arbeit selbständig und ohne fremde Hilfe von mir angefertigt wurde, und ich bis auf die in der Arbeit angegebenen Hilfsmittel keine anderen verwendet habe. Alle Stellen, die wörtlich oder sinngemäß anderen Schriften entnommen wurden, sind als solche kenntlich gemacht.

Heilbronn, 15.02.2011

_____

Rudolf Erich Robinigg