Erweiterung der HL7 Clinical Document Architecture um Prozessstatusinformation

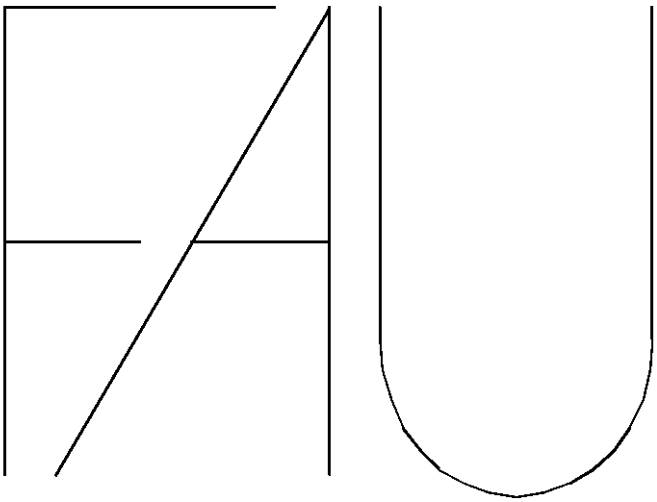
Emil Taralov

Diplomarbeit

Lehrstuhl für Informatik 6
(Datenbanksysteme)

Institut für Informatik
Technische Fakultät

Friedrich-Alexander-
Universität
Erlangen-Nürnberg



Erweiterung der HL7 Clinical Document Architecture um Prozessstatusinformation

Diplomarbeit in Informatik

erstellt von

Emil Taralov

geboren am 05.05.1982 in Burgas, Bulgarien

angefertigt am

Lehrstuhl für Informatik 6 (Datenbanksysteme)
Institut für Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

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

Beginn der Arbeit: ***01.08.2008***
Abgabe der Arbeit: ***02.02.2009***

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

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

Erlangen, den 2. Februar 2009 _____

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Viele Bereiche des täglichen Lebens werden von der Informations- und Kommunikationtechnik überdimensional beeinflusst. In zunehmenden Maße prägt der Einsatz technischer Entwicklungen das Informations- und Kommunikationsumfeld im Gesundheitswesen. Verstärkt wird in die gezielte Entwicklung klinisch-technologischer Konzepte investiert. Durch die Kombination von Standard- und Individuallösungen wird versucht die Krankenhäuser als Gesundheitszentren weiter zu entwickeln, wobei auch andere Institutionen des Gesundheitswesens und niedergelassene Ärzte ebenfalls in dieses „Netzwerk“ miteinbezogen werden [Pflüglmayer01]. Die IT-Unterstützung institutionsübergreifender Behandlungsprozesse und ein weiterhin lückenloser Informationsfluss patientenbezogener Daten über die Institutionsgrenzen hinweg, stellt ein zentrales Konzept dieser Versorgungskontinuität dar. Voraussetzung dabei ist die mögliche Erweiterbarkeit und Integration in bestehende Systeme im Gesundheitsbereich sowie die Wiederverwendbarkeit der gespeicherten Informationen. Die semantische Interoperabilität der ausgetauschten Daten ist ebenfalls ein wichtiger Faktor. Diese soll eine technische und anwendungslogische Kommunikation für die Systeme im Gesundheitsbereich gewährleisten. In der Sphäre der medizinischen Informatik ist zum Informationsaustausch zwischen unterschiedlichen Systemen in diesem Bereich mit „Health Level 7“ (HL7) bereits ein aufstrebender Standard vorhanden. Die neue „Health Level 7“ Version 3 (HL7 V3) repräsentiert einen wichtigen Schritt hinsichtlich der Erstellung medizinischer Nachrichten und ist Basis für die „Clinical Document Architecture“ (CDA) [Hinchley07]. Mit CDA wird eine flexible Dokumentenarchitektur als ein Standard zur Strukturierung, zum Inhalt und zum Austausch klinischer Dokumente zur Verfügung gestellt, welche die oben erwähnten Ansprüche zur Interoperabilität der ausgetauschten Daten adressiert [Flemming08].

1.1 Motivation

Der durchschnittliche Patient muss im Laufe einer Behandlung unter Umständen zahlreiche Institutionen durchlaufen (vgl. **Abbildung 1**).

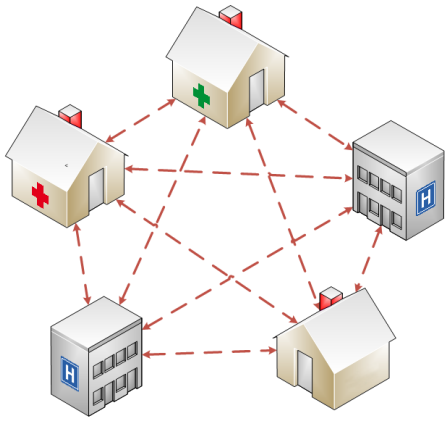


Abbildung 1: Im Laufe einer Behandlung durchläuft der Patient mehrere Institutionen (in Anlehnung an [HL7CH])

Während dessen werden zahlreiche medizinische Dokumente in Bezug auf diese Behandlung generiert. Heute werden diese Dokumente grundsätzlich mittels Briefen, Fax oder E-Mail ausgetauscht. In vielen Fällen wird der Patient selbst als Kurier eingesetzt, indem er das Dokument (z.B. Austrittbericht, Röntgenbild, etc.) nach Erbringung einer Leistung von der erstellenden Institution (z.B. Radiologie) beim nächsten Termin in einer anderen Institution persönlich mitbringt (z.B. Arztpraxis, Facharzt) (vgl. **Abbildung 2**).

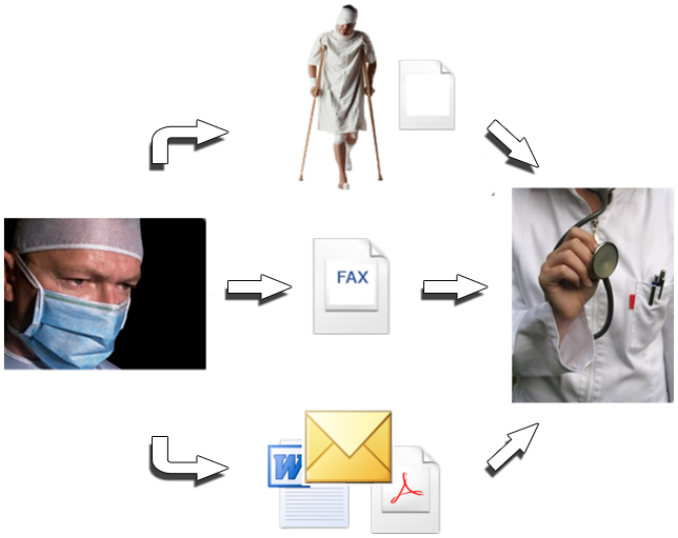


Abbildung 2: Dokumentenaustausch unter Leistungserbringern erfolgt hauptsächlich in Form von Briefen, Fax oder E-Mail (in Anlehnung an [HL7CH])

Für eine rechtzeitige, patientenorientierte Betreuung wird eine rasche Übermittlung relevanter medizinischer Patient-Informationen zwischen den einzelnen Institutionen benötigt. Studien zeigen, dass in der Medizin häufig Fehler auftreten, die zu einem großen Teil vermeidbar wären. Als Ursachen lassen sich mangelnde Verfügbarkeit von Informationen, schlechte Kommunikation und Koordination ausmachen. Beispielsweise sind 18% der Medikationsfehler durch fehlende patientenbezogene Informationen und 29% durch fehlendes medizinisches Wissen verursacht [Leape95]. Die Notwendigkeit, medizinische Dokumente zwischen den Institutionen elektronisch auszutauschen ist nicht mehr zu übersehen. Heute erfolgt dieser Austausch immer noch zu einem grossen Teil auf Papier. Im Anbetracht der technischen Möglichkeiten im IT-Bereich liegt in dieser Hinsicht noch ein großes ungenutztes Potenzial.

1.2 Problemdarstellung

In der Praxis wird ein Behandlungsprozess nicht zentral überwacht, sondern der Patient wird von einem Laufzettel begleitet, der die notwendigen Informationen zu Prozess und Status enthält. Der isolierte Betrieb verschiedener Informationssysteme der einzelnen medizinischen Institutionen führt häufig zu Medienbrüchen und zur einer mangelnden Kommunikation. Die „Zusammenarbeit“ dieser Systeme, die auch als Interoperabilität bezeichnet wird, gewinnt deshalb immer mehr an Bedeutung. Die notwendige

Interoperabilität wird heute vor allem über Mechanismen zum Datenaustausch hergestellt. Nach [HL7CH] können vier verschiedene Levels der Interoperabilität definiert werden (vgl. **Tabelle 1**). Jede dieser Kategorien erfordert unterschiedliche technische Voraussetzungen und bietet verschiedene Nutzenpotenziale beim Datenaustausch.

Level	Daten	Beispiele
1	Daten, die nicht elektronisch sind	Papier, Brief
2	Daten, die elektronisch übertragbar sind	E-Mail, Fax
3	Daten, die maschinenstrukturierbar sind	indexierte Dokumente
4	Daten, die maschineninterpretierbar sind	Automatisierte Übertragung von codierten Laborresultaten eines externen Labors in die Fallakte eines KIS.

Tabelle 1: Interoperabilitätskategorien im Gesundheitswesen beim Datenaustausch (in Anlehnung an [HL7CH])

Die CDA basiert auf XML und unterscheidet drei inhaltliche Abstraktionsstufen. Diese repräsentieren die unterschiedliche Feinheit und Granularität der wiedergegebenen klinischen Informationen und maschinenauswertbaren Markups in einem HL7 CDA Dokument. Ebene 1 erfasst menschenlesbaren Inhalt und Strukturierungselemente (z.B. Paragraphen, Überschriften etc.). Die nächsten zwei Ebenen sind für eine weitergehende semantische Annotation der medizinischen Dokumentinhalte zuständig. Die CDA erfüllt bereits auf Ebene 1 die dritte Interoperabilitätskategorie. Prozessbegleitende Informationen zu Arbeitsablauf (Workflow) und Prozessstatus fehlen aber CDA Dokumenten. Es stellt sich die Frage, inwieweit sich CDA über den reinen Dokumentationscharakter hinaus als Basis eines Gesundheitsinformationssystems nutzen lässt, so dass man von der Plattformunabhängigkeit und Evolutionsfähigkeit des CDA-Konzeptes profitieren kann.

1.3 Zielsetzung der Diplomarbeit

Zur Unterstützung von verteilten Behandlungsprozessen auf Basis von CDA Dokumenten, soll eine Möglichkeit erarbeitet werden, Prozessstatusinformationen übermitteln zu können. Zwei Ansätze sind vergleichend zu evaluieren: einerseits könnte der CDA Header aller an der Geschäftsprozessinstanz beteiligten Dokumente erweitert werden, andererseits könnte ein zusätzliches prozessindividuelles CDA Level 1 Dokument als Bezugsobjekt der Geschäftsprozessinstanz instrumentiert werden. Das Schema der Prozessstatusinformation soll sich an etablierten Standards orientieren, wie z.B. BPEL, XPDL oder ebXML.

Die erarbeitete Vorgehensweise soll dann als Basis bei der Entwicklung einer IT-Infrastrukturplattform dienen. Damit die Plattform eine möglichst gute Unterstützung bietet, aber die realen Abläufe wenigstmöglich einschränkt, sollte sie sich an diesen

Abläufe orientieren. Das kann erreicht werden, indem diese den Behandlungsprozess begleiten, wodurch der Patient als zentraler Bezugspunkt in den Fokus rückt.

2 Modellierung und Entwicklungsvorgang

In diesem Kapitel wird der Entwicklungsvorgang einer Workflow-Management-Anwendung zur Unterstützung von verteilten Behandlungsprozessen vorgestellt und die Workflow-Modellierung und ihre Position im Entwicklungsvorgang präzisiert.
Die Grundlage der Entwicklung der Zielanwendung stellt die Ist-Analyse der klinisch-medizinischen Anwendungsdomäne und ihren spezifischen Zwecksetzung und Kriterien bzw. Constraints, die berücksichtigt werden müssen dar. Die Ergebnisse dieser Analyse sind einzelne Arbeitselemente, wie zum Beispiel (externe diagnostische Maßnahme ergreifen oder externe therapeutische Maßnahme ergreifen). Diese werden im Rahmen der Arbeitssynthese zu möglichen Workflows zusammengefasst.
In den folgenden drei Kapiteln wird dann das fachliche Lösungskonzept für das geplante Informationssystem in zwei Teilschritten erarbeitet. Zuerst wird die Lösung ausschließlich „problemorientiert“ und somit unabhängig von einer spezifischen Art von Basissoftware sowie von einer spezifischen Lösungsarchitektur spezifiziert. Es werden die funktionalen Anforderungen an das zu entwickelnde System, als auch die nichtfunktionalen Anforderungen wie Offenheit, Erweiterbarkeit und Zuverlässigkeit erstellt. Das Kapitel 5 beginnt mit einer Vorstellung inklusive Evaluation existierender Open-Source-Frameworks, die als Basissoftware eingesetzt werden. Erst im Kapitel 6, im Rahmen der technischen Anforderungsanalyse wird die Entscheidung über einen bestimmten Anwendungssystemtyp, wie zum Beispiel einer Workflow-Management-Anwendung getroffen und die Systembausteine und die Grobarchitektur vorgestellt.
Im Anschluss an die Ausarbeitung wird die Phase des Systementwurfs beschrieben. Zuerst werden die ausgewählten Arbeitsabläufe durch den Einsatz von Werkzeugen modelliert, IT-bezogen vervollständigt und präzisiert. Bis zu dieser Phase der Entwicklung spielt die geplante Verwendung eines Workflow-Management-Systems noch keine Rolle. Im zweiten Teil des Kapitels stehen die Modularisierung und der Entwurf der Gesamtarchitektur inklusive der vom System angebotenen Schnittstellen im Vordergrund. Das Testkonzept und die Festlegung der Testfälle für den Systemtest in der Konfigurationsphase runden dieses Kapitel ab.
Das folgende achte Kapitel hat die Aufgabe die Implementierungsphase bei der Entwicklung zu beschreiben. In dieser Phase ist letztlich die Ausführbarkeit durch das Workflow-Management-System herzustellen. Damit ist die technische Umsetzung der im Systementwurf spezifizierten Module und die Workflow-Implementierung gemeint. Die Arbeitsabläufe wurden innerhalb des Systementwurfs erstellt und bilden hier den Ausgangspunkt für die Spezifikation von Workflows. Es gibt an dieser Stelle keine Möglichkeit, den Workflow unabhängig von dem Workflow-Management-System zu spezifizieren. Im Gegenteil, die Entscheidung für ein Workflow-Management-System zwingt auch zum Gebrauch seiner speziellen Workflow-Sprache. Mit den Mitteln dieser Workflow-Sprache wird dann das Workflow-Modell formuliert und die dahinter liegende Logik implementiert.
Die letzte Phase der Entwicklung - die Systemkonfiguration erfolgt in Kapitel 9. Dabei werden die einzelnen Komponenten zu einem funktionsfähigen System verbunden und

konfiguriert. Neben dieser Aufgabe, erfolgt in dieser Phase auch der Systemtest, bei dem das System in konkreten Szenarien und unter Betriebsbedingungen getestet wird.

3 Der diagnostisch-therapeutische Zyklus in der Medizin

In den letzten Jahren ist der medizinische Sektor in Deutschland in einem kontinuierlichen Wandel. Grund dafür ist ein permanenter Kostendruck und eine sich ständig wandelnde Gesetzgebung. Aus informationstechnischer Sicht könnte ein Kostensenkungsprozess durch das Konzept der Prozessorientierung und Optimierung der Abläufe unterstützt werden. Die Adaption des Prozessgedankens in den Bereich der Medizin wird in der Literatur als „klinischer Pfad“ bezeichnet. Bei klinischen Pfaden handelt es sich vereinfacht ausgedrückt um medizinische Prozesse. Sie sollen die Arbeitsabläufe im Gesundheitsbereich standardisieren und den optimalen Weg eines bestimmten Patiententyps mit seinen entscheidenden diagnostischen sowie therapeutischen Leistungen und seiner zeitlichen Abfolge im Krankenhaus beschreiben [Müller07].

Es ist wichtig für den klinischen Routinebetrieb, dass den einzelnen Akteuren des klinischen Pfades, eine umfassende Entscheidungs- und Handlungsfreiheit eingeräumt wird, um auf unvorhergesehe Ereignisse, wie z.B. Notfälle, reagieren zu können [Dadam00]. Im Vergleich dazu sind die betriebswirtschaftlichen Abläufe über mathematische Logik bis ins kleinste Detail konzertiert und nach Plan durchgeführt (Fließband, Büroarbeit, etc.). Dazu noch kann die fortwährende Zunahme des medizinischen Wissens schwer in statischen Modellen festgeschrieben werden. Auf Grund dessen können Workflow-Management-Systeme nicht ohne größere Umstände auf ein Krankenhaus übertragen werden. In diesem Zusammenhang müssen, zwecks institutionsübergreifender Prozessunterstützung, möglichst generische Prozessschritte spezifiziert und modelliert werden.

In diesem Abschnitt beschäftigt sich die Arbeit mit der Untersuchung der Behandlungsprozesse und ihren spezifischen Prozessschritten. In diesem Kontext wird versucht ein möglichst generisches Modell abzuleiten, welches dann als Grundlage der Entwicklung der prozessunterstützenden Anwendung dienen kann. Zu Beginn stellt man sich die allgemeinen Fragen: Was passiert eigentlich im Krankenhaus? Welcher medizinische Behandlungsprozess kann als allgemein gültig bezeichnet werden?. Diese wichtigen Fragen werden mit der Vorstellung des diagnostisch-therapeutischen Prozesses in der Medizin beantwortet.

Jeder Behandlungsprozess beginnt mit der Sammlung von patientenbezogenen Informationen. Der Patient kommt in ein Krankenhaus und bringt seine Anamnese mit. Er wird vom Arzt zu den Symptomen befragt und es erfolgt z.B. eine körperliche Untersuchung (vgl. **Abbildung 3**). Nach der Sammlung dieser wichtigen Informationen erfolgt die Entscheidungsfindung. Der Arzt muss sich anhand der gesammelten Informationen für die weiteren Maßnahmen im Behandlungsprozess entscheiden. Der Nutzen von Information und Wissen für das ärztliche Handeln bei der klinischen Entscheidungsfindung und Problemlösung unterscheidet sich in Diagnostik und Therapie. Bei der Diagnostik wird anhand von mehreren potentiellen Möglichkeiten, die auf anfänglichen Befunden basieren, die endgültige Diagnose eingegrenzt. Somit wird bis zur abschließenden Diagnose ein Prozess der Komplexitätsreduktion betrieben. Als Beispiel könnte nach der körperlichen Untersuchung eine Blutentnahme und

Laboruntersuchung als diagnostische Maßnahme eingeleitet werden. Damit die Unsicherheit bei dieser Entscheidung reduziert wird, käme auch eine Röntgenuntersuchung im Betracht. Insofern ist der diagnostische Prozess durch Phasen der Unsicherheit charakterisiert [Kaltenborn99].

Abbildung 3: Informationssammlung als Teil des diagnostisch-therapeutischen Prozesses in der Medizin (in Anlehnung an [Prokosch08])

Bei der Erstellung des Therapieplans sind in der Regel, wenn auch nicht immer, derartige Unsicherheitsphasen bereits beseitigt und die zusätzlichen Informationen aus den verschiedenen Informationsquellen müssen vorrangig den neuesten therapeutischen Wissensstand garantieren. Der Prozess der Unsicherheitsreduktion könnte aber sowohl lokal in einer medizinischen Organisationseinheit als auch verteilt zwischen mehreren Organisationen verlaufen und kann generisch mit dem diagnostisch-therapeutischen Zyklus in **Abbildung 4** geschildert werden.

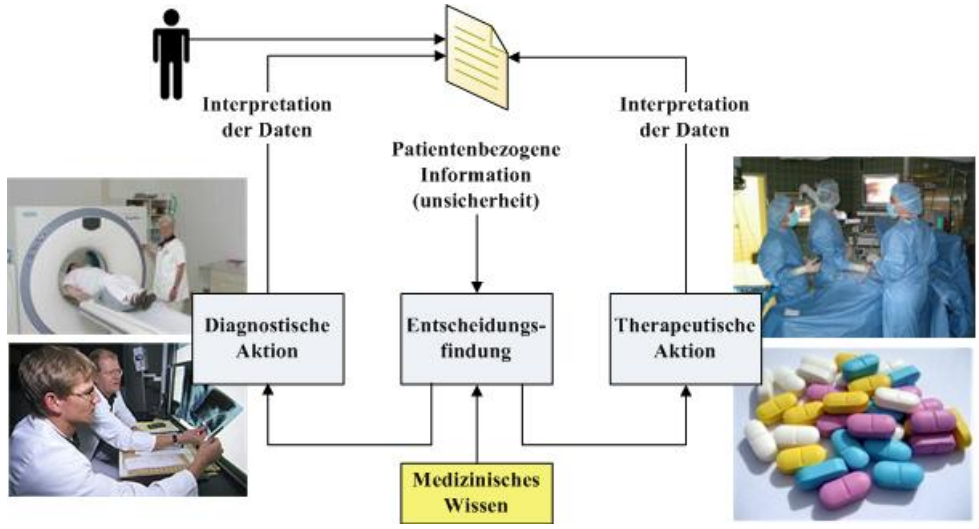


Abbildung 4: Diagnostisch-therapeutischer Zyklus in der Medizin (in Anlehnung an [Prokosch08]

4 Anforderungsanalyse

Ziel dieses Kapitels ist es, die wichtigsten Anforderungen an das System im Hinblick auf seine Funktions-, Effizienz-, Qualitätsmerkmale und auf die gegebenen technischen Rahmenbedingungen hinreichend, vollständig und präzise zu beschreiben.

4.1 Zielbestimmung des Systems

Der Anwendungsbereich einer IT-Plattform, die verteilte diagnostisch-therapeutische Prozesse zentral überwacht und unterstützt, stellt weitgehende Anforderungen an die Fähigkeit, die zahlreichen medizinischen Dokumente auf dem Behandlungspfad eines Patienten in Form von HL7 CDA Dokumenten zu übermitteln. Diese sollen die heutzutage ausgetauschten Dokumente hauptsächlich in Form von Briefen auf Papier, Fax oder E-Mail ersetzen. Die Realisierung einer solchen Plattform stellt umfassende Anforderungen, die im Rahmen dieser Arbeit in funktionalen und nichtfunktionalen eingeteilt werden. Im Folgenden werden die funktionalen Anforderungen im Bezug auf die angebotenen Funktionalitäten beschrieben und im Anschluss die gestellte Bedingungen an die Funktionsausführung, zum Beispiel Zuverlässigkeit und an die Offenheit des Systems betrachtet.

4.2 Funktionale Anforderungen an das Zielsystem

Die funktionalen Anforderungen an die Workflow-Management-Anwendung sind einerseits durch die Workflow-Sprache und deren Ausdruckskraft und andererseits durch Angaben und Spezifikationen bezüglich der Benutzerschnittstellen bestimmt. Aufgrund dieser Aufteilung werden diese Anforderungen im Bezug auf das zu entwickelnden System in drei Bereiche gegliedert und im Folgenden detailiert beschrieben.

4.2.1 Das Workflow-Modell des diagnostisch-therapeutischen Prozesses

Da das Zielsystem die Unterstützung der verteilten diagnostisch-therapeutischen Arbeitsabläufe zum Ziel hat, stellt die korrekte Überführung dieser Abläufe in ein Workflow-Modell sowie die Implementierung dieses Modells in einen Workflow, die zentrale technische Anforderung dar. Da der Austausch von medizinischen Dokumenten als Hauptfunktionalität im Fokus der Entwicklung steht, ist die Anforderung an die Verwaltung dieser Informationen sowie die Bewerkstelligung des Dokumentenflusses zwischen den Workflows und externen Systemen ein wichtiger Aspekt. So hat der Workflow, Funktionen zur Verwaltung und Austausch von Dokumenten bereitzustellen. Die komplexen Behandlungspfade beschränken sich selten auf Ausführung in einer einzelnen Institution wie z.B. einer Praxis. Meist sind mehrere Institutionen in diesem verteilten Prozess beteiligt. Sieht man auf jeder beteiligten Institution eine Workflow-Engine zur Ausführung vor, die einen verteilten Behandlungsprozess unterstützt, stellt sich neben der Kommunikation noch das Problem der Synchronisation zwischen den

verschiedenen Workflow-Engines dar. Diese Tatsache erfordert, dass Nachrichten mit Synchronisationsdaten zwischen den einzelnen Prozessteilnehmern ausgetauscht werden. Derartige Nachrichten enthalten Kontextänderungen, wie Events, Variablen, etc. die in der Workflow-Engine des Senders stattgefunden haben. Aufgrund dieser Information ist es möglich, dass die Nachrichten beispielweise Transitionen schalten oder Workflow-Instanzen in der Workflow-Engine des Empfängers starten oder beenden.
Ein ebenfalls wichtiger Punkt in diesem Kontext ist die Nachverfolgbarkeit des verteilten Prozesses während der Behandlung. Dies impliziert den Austausch von Informationen bezüglich der Prozessteilnehmer zwecks Historienbildung, welche die Nachvollziehbarkeit der Behandlungsabläufe ermöglicht.
Dabei fordert eine solche Architektur, neben dem Dokumentenfluss zudem noch die Bewerkstelligung eines Kontrolldatenflusses zwischen den Workflow-Engines.

4.2.2 Umgebung des Zielsystems

Medizinische Informationssysteme finden zunehmend in allen Bereichen des Gesundheitswesens ihren Einsatz. Funktionalität und Angemessenheit schwanken aber erheblich. In den Krankenhäusern existieren schon technische Systeme, auch Krankenhausinformationssystem (KIS) genannt, welche alle informationsverarbeitenden Prozesse und die an ihnen beteiligten menschlichen und maschinellen Handlungsträger in ihrer informationsverarbeitenden Rolle umfassen. Neben den oben genannten Systemen sind die sogenannten klinischen Unterstützungssysteme besonders relevant für die medizinische Qualität im Krankenhaus. Dies sind vor allem Systeme, die den Arzt des jeweiligen Fachgebietes bei der Beobachtung, Diagnostik und Therapie direkt unterstützen und daher meist medizinisch-technischen Charakter haben. Sie sind oft aufgrund lokaler Bedürfnisse von Fachabteilungen des Krankenhauses entstanden. Klinische Unterstützungssysteme sollen primär die funktionsdiagnostischen Bereiche eines Krankenhauses unterstützen. Breite Anwendung finden insbesondere radiologische Informationssysteme (RIS), die sowohl die medizinisch-technischen als auch administrativen Funktionen der Radiologie-Abteilungen unterstützen [Pflüglmayer01]. Die Systeme können entweder als Einzelsysteme oder in ein KIS integriert sein. Darüber hinaus sind aus Sicht der niedergelassenen Ärzte, die Praxis-Verwaltungsmanagementsysteme zu erwähnen.
Das System zur Unterstützung von verteilten diagnostisch-therapeutischen Prozessen wird als eine Erweiterung der Informationssysteme im Umfeld der betrachteten medizinischen Organisationseinheiten geplant. Aufgrund des breiten Spektrums der existierenden Informationssysteme in den einzelnen Organisationseinheiten wird in dieser Arbeit ein generischer Begriff eingeführt, der die Gesamtheit dieser Systeme in einer medizinischen Organisationseinheit umfasst: das „Healthcare Information System“ (HCIS)[1]. Weiterhin wird in der vorliegenden Arbeit das zu entwickelnde System mit dem Begriff „Distributed Medical Process System“ (DMPS) bezeichnet.
Mit der Einführung neuer IuK-Techniken sind jedoch meist Probleme verbunden, da oft heterogene Software- und Hardwareumgebungen vorhanden sind. Die einfache

[1] HIS hätte Konflikt mit Hospital IS (englisch) ≡ KIS (deutsch)

Anbindung des DMPS an die heterogenen HCI-Systeme erfordert, dass Schnittstellen zur Kommunikation mit diesen Systemen bereitgestellt werden (vgl. **Abbildung 5**). Man strebt für das System eine möglichst lose Kopplung an, also ein System, das weitgehend unabhängig ist und mit den anderen Systemen nur über möglichst schmale, wohl definierte Schnittstellen kommuniziert.

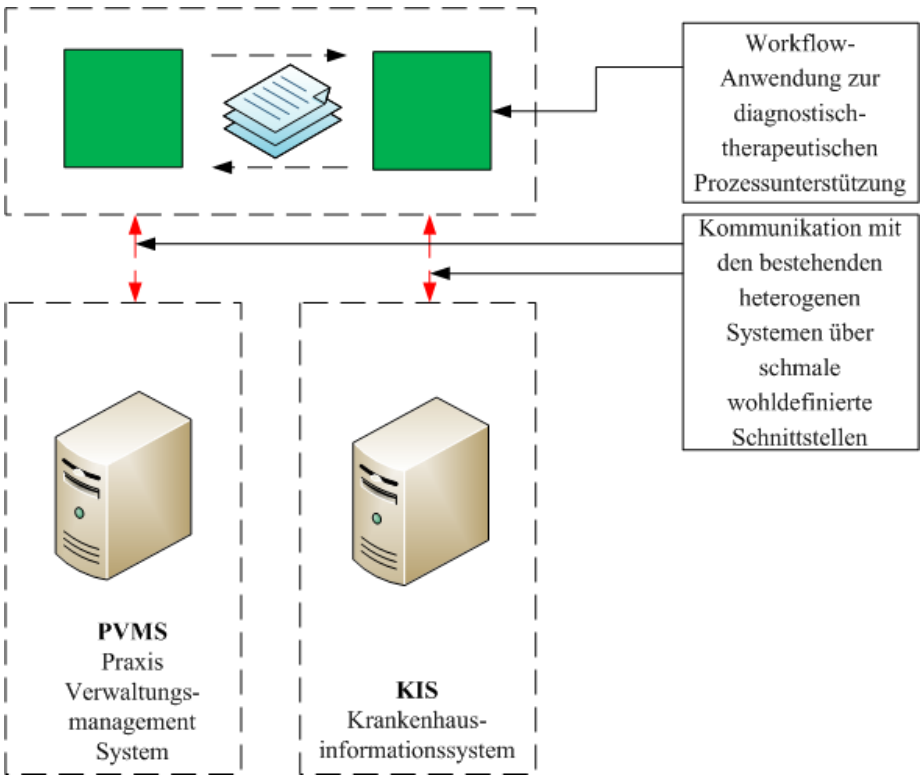


Abbildung 5: Das Zielsystem soll über Schnittstellen mit den bestehenden HCI-Systemen kommunizieren

Das Anwendungsfalldiagramm in **Abbildung 6** stellt das äußerlich erkennbare Systemverhalten aus der Sicht des Anwenders (das lokale HCIS und externe Benutzer) und die Beziehungen zwischen Akteuren und Anwendungsfällen dar. Jeder dieser Anwendungsfälle besteht aus mehreren eng zusammenhängenden Aufgaben, die von einem Aktor in Kooperation mit dem Zielsystem durchgeführt werden. Aus Sicht der HCIS ergibt sich die Anforderung den Ablauf der Prozesse in der unterstützenden Workflow-Anwendung zu steuern. Zum einen hat das Zielsystem die Initiierung externer Behandlungen zu ermöglichen. Während dessen Verlauf, ist der Status der Behandlungen eine wichtige Information, die vom System zum Abruf bereitgestellt werden muss. Zum anderen ist die Beendigung von einzelnen externen Behandlungen sowie des ganzen Prozesses als Funktionalität anzubieten. Andererseits soll das HCIS von dem Zielsystem für den Fall, dass Fehlerzustände auftreten oder externe Dokumente empfangen wurde, benachrichtigt werden. Die Interaktion soll also bidirektional erfolgen. Deswegen sind neben den Schnittstellen, die von dem Zielsystem

bereitzustellen sind, noch alle externen HCIS-Schnittstellen zu beschreiben, die gegenüber dem Zielsystem als Informationssenken auftreten können.

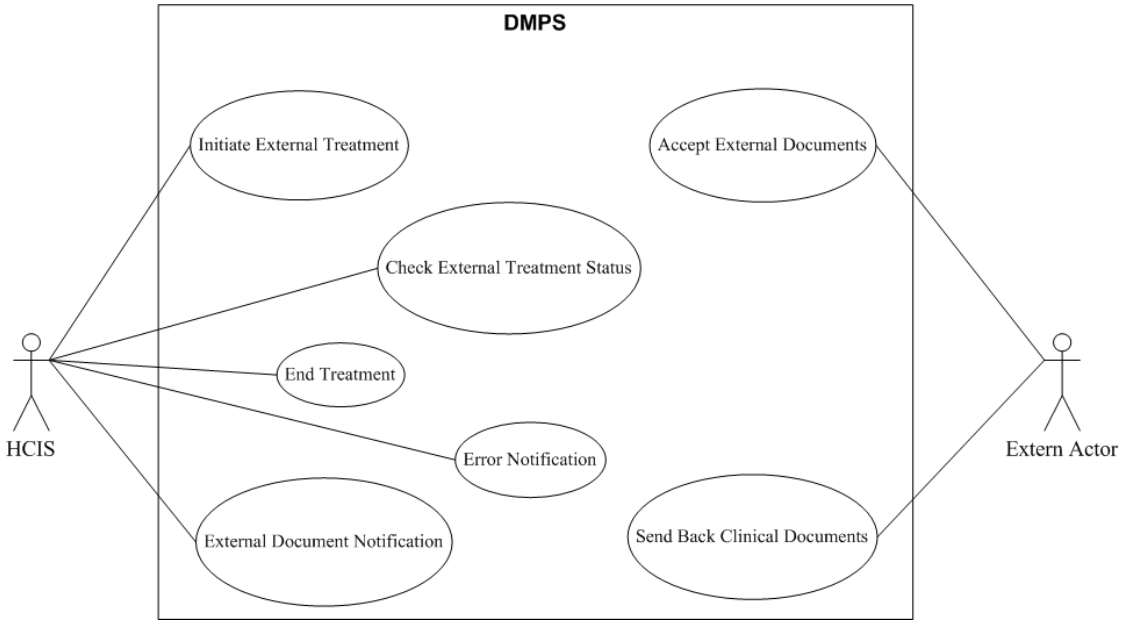


Abbildung 6: Die Anwendungsfälle beschreiben typische Situationen, in denen das Zielsystem verwendet wird

4.2.3 Funktionalität gegenüber externen Benutzern

In diesem Abschnitt werden die wesentlichen Funktionen, die das Zielsystem in Hinsicht auf die externen Benutzer erfüllen soll, kurz beschrieben. Zu Beginn werden zwei Begriffe bezüglich verteilter Behandlungsprozesse eingeführt. Man kann die Struktur der Abwicklung einer verteilten Behandlung in Hinsicht auf die teilnehmenden Institutionen als eine Hierarchie betrachten: eine Anordnung von Organisationseinheiten, die bezüglich der Teilnahmereihenfolge an der Behandlung einander über- bzw. untergeordnet sind. Dabei bezeichnen die „downstream“ und „upstream“ Begriffe die Flussrichtungen der Daten (Dokumenten) in dieser Hierarchie. Als „downstream Participant“ in einer Behandlung wird dann der nachgeschaltete Teilnehmer und als „upstream Participant“ den vorgeschalteten Teilnehmer identifiziert (vgl. **Abbildung 7**).

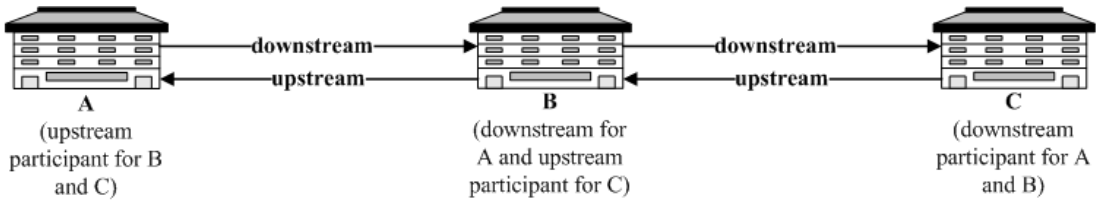


Abbildung 7: Behandlungsinitiierung und Ablauf zwischen „upstream“ und „downstream“ Teilnehmer

Wie schon in den früheren Kapiteln erwähnt, soll das Zielsystem die verteilten diagnostisch-therapeutischen Prozesse unterstützen indem die Initiierung von externen Behandlungsprozessen ermöglicht wird. Die externen Benutzer des Systems sollen in der Lage sein, medizinische Dokumente mit diesen auszutauschen und so externe Behandlungen zu initiieren und abzuwickeln (vgl. **Abbildung 6**). Als Beispiel wird noch mal die **Abbildung 7** in Betracht gezogen. Die Organisationseinheit A schickt den „downstream participant" B medizinische Dokumente. Der „downstream participant" B bietet Funktionalitäten zum Austausch von Dokumente und aus Sicht von A wurde in B eine externe Behandlung eingeleitet. An dieser Stelle ist es wichtig abzugrenzen, wer einen verteilten diagnostisch-therapeutischen Prozess initiiert.
Einerseits interagieren HCI-Systeme, die das DMPS integriert haben. Das HCIS triggert dann die Initiierung einer externen Behandlung in dem lokalen DMPS, welches medizinische Dokumente den „downstream participant" schickt. Ein mögliches Szenarium ist in der folgenden Abbildung geschildert.

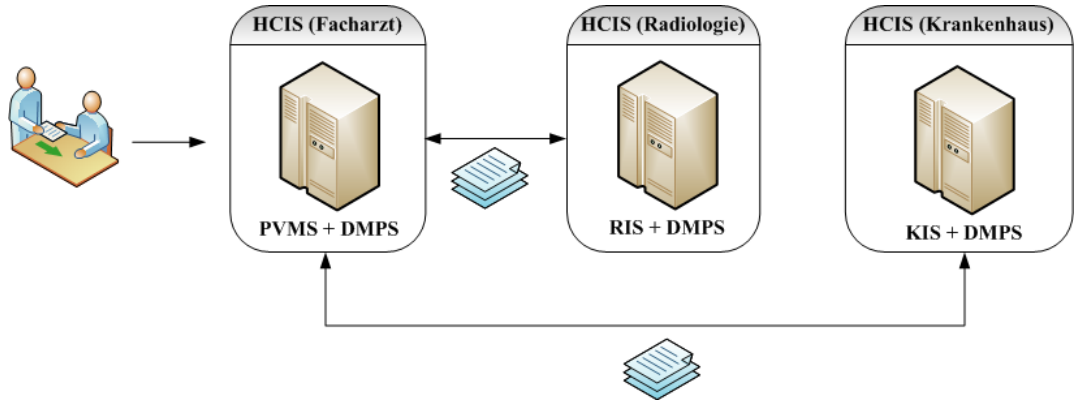


Abbildung 8: Szenarium einer verteilten Behandlung. Die Interaktion erfolgt ausschließlich zwischen DMP-Systeme

Ein Patient kommt direkt zum Facharzt. Während der lokalen Behandlung, benötigt der Patient eine externe Behandlung in dem funktionsdiagnostischen Bereich eines Krankenhauses (Radiologie). Das HCIS des Facharztes triggert die Initiierung der externen Behandlung im lokalen DMPS, welches dann die medizinischen Dokumente dem HCIS der Radiologie übergibt. Die funktionsdiagnostischen Bereiche dieses Krankenhauses (z.B. Radiologie) erstellen täglich eine große Zahl an Befunde. Ein DMPS kann in diesem Anwendungsbereich als eine Erweiterung der radiologischen Informationssysteme (RIS), die diagnostischen Arbeitsabläufe unterstützen und darüberhinaus die Befunddokumentationübermittlung beschleunigen und vereinfachen. Der Facharzt wartet bis die Befunddokumentation von der Radiologie dem HCIS zurückgesendet wird. Aufgrund dieser Dokumentation entscheidet der Arzt, dass eine externe Therapie im Krankenhaus benötigt wird. Es wird eine weitere externe Behandlung eingeleitet, indem medizinische Dokumente dem HCIS im Krankenhaus übermittelt werden. Nachdem die Behandlung im Krankenhaus ebenfalls beendet wird, sendet das HCIS die Resultate an dem Facharzt.

Andererseits soll es möglich sein, dass die Interaktion auch zwischen non-DMPS und DMPS erfolgt, d.h. dass ein non-DMPS einem DMPS medizinische Dokumente übergeben kann. Ein solcher Ablauf ist mit einem zweiten Szenarium in **Abbildung 9** abgebildet. In diesem Fall wird der verteilte diagnostisch-therapeutische Prozess von einem non-DMPS initiiert indem ein Allgemeinarzt einen Arztbrief einem Facharzt zur weiteren Behandlung sendet. Dies könnte zum Beispiel mittels eines Webportals beim Facharzt geschehen. Das DMPS soll dann die Dokumente empfangen und an dem HCIS weiterleiten. Der Rest des Szenariums läuft genauso wie im ersten vorgestellten Fall (vgl. **Abbildung 8**) ab.

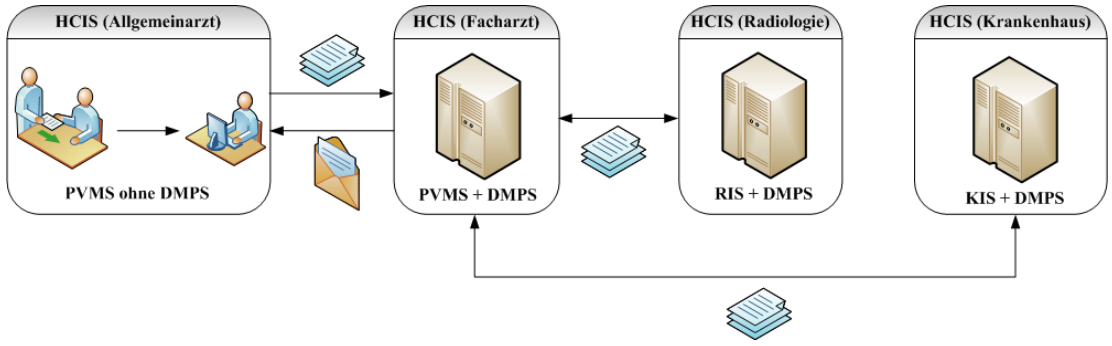


Abbildung 9: Szenarium einer verteilten Behandlung mit einem non-DMPS, das den Behandlungsprozess initiiert

4.3 Nicht-funktionale Anforderungen an das Zielsystem

Neben der eingeführten Funktionalität, muss die Workflow-Management-Anwendung noch verschiedensten nichtfunktionalen Anforderungen genügen. Eine der wichtigsten Anforderung für derartige Software-Systeme ist diese der Offenheit [Jablonski97]. Dieser Begriff wurde vom „Institute of Electrical and Electronics Engineers“ (IEEE) definiert als „ein System, das in ausreichendem Maße offengelegte Spezifikationen für Schnittstellen implementiert, damit entsprechend entwickelte Anwendungssoftware:

- auf eine Vielzahl verschiedener Systeme portiert werden kann,
- mit anderen Anwendungen lokal und entfernt interoperable ist,
- mit Benutzern in einer Art interagiert, die das Wechseln der Benutzer zwischen den Systemen erleichtert.“ [POSIX 92]

Die Umsetzung der Offenheit-Eigenschaften sichert eine Wiederverwendung des Systems, d.h. ein System, das in veränderten Umgebungen weiterhin arbeitsfähig und in unterschiedlichen Infrastrukturen integrierbar ist. Dabei entsteht Nutzen sowohl für den Anwender des Systems als auch für den Entwickler. Einerseits ergibt sich für den Anwender die Möglichkeit die bestehenden und vertrauten Systeme und Umgebungen beizubehalten sowie einen Schutz der geleisteten Investition. Andererseits bedeutet die Offenheit für den Entwickler die einfache Portabilität der Anwendung nach

unterschiedlichen Anforderungen und Gegebenheiten bei geringfügigem finanziellem Aufwand. Aus Sicht der zu entwickelnden Anwendung, die in den unterschiedlichsten und heterogenen HCIS integriert werden soll und eine möglichst lose Kopplung über möglichst schmale, wohl definierte Schnittstellen bereitstellt, stellt der Begriff der Offenheit eines Systems wichtige nicht-funktionalen Anforderungen an das Zielsystem.
Die Erweiterbarkeit des Systems ist eine weitere allgemeingültige Anforderung für Software-Anwendungen. Auch in dieser Arbeit sind bei der Entwicklung, mögliche Erweiterungen der Plattform in Betracht zu ziehen. Als Erweiterung wird an dieser Stelle der zukünftige Funktionsausbau hinsichtlich der „Future Work"-Vorschläge (vgl. Kapitel 11) oder eine mögliche Änderung der Funktionalität angedacht. Der zentrale Punkt ist es Mechanismen zur Erweiterung des Systems bei minimalem Einfluss auf die bestehende Funktionalität bereitzustellen.
Eine weitere nichtfunktionale Anforderung, die man unter den Begriff der Zuverlässigkeit des Systems einordnen kann, ist die Unterstützung von Recovery-Mechanismen. Zum einen ist die Persistenzhaltung der in der Workflow-Management-Anwendung vorhandenen und relevanten Daten ein wichtiger Mechanismus zur Wiederverwendung von Informationen und somit zur Sicherung der Zuverlässigkeit des Systems. Insbesondere nach einem Systemfehler (z.B. Absturz des Workflow-Management-Systems) sollte der Recovery-Mechanismus mit der Hilfe von der Persistenzhaltung in der Lage sein, einen konsistenten Workflow-Zustand zu rekonstruieren und die Workflow-Ausführung fortzusetzen. Dies wird häufig dadurch erreicht, indem die Workflow-Anwendung mit einem DBMS gekoppelt wird.
Ein weiterer Mechanismus ist die Möglichkeit zur Wiederherstellung nach semantischen Fehlern. Ein solcher Fehler wird durch ein logisches Fehlschlagen einer Aktivität verursacht. Ein solcher Fall wäre die Dokumentenübertragung trotz momentaner Nichtverfügbarkeit dieser Dokumente. Nach semantischen Fehlern könnte der DMPS-Workflow in einen inkonsistenten Zustand überführt werden. Der Wiederherstellungsmechanismus soll in solchen Situationen Unterstützung bereitstellen, indem das System möglichst automatisch einen konsistenten Zustand herzustellen versucht. Dies könnte durch Kompensation oder Ausführung von Ersatzaktivitäten geschehen.
An dieser Stelle ist es wichtig abzugrenzen, wer bei einem Informationsaustausch zwischen zwei oder mehreren DMP-Systeme identifiziert werden soll. In dieser Hinsicht ist die Patientenidentifikation nicht Teil dieser Arbeit. Das Zielsystem soll lediglich die medizinischen Organisationseinheiten sowie die externen laufenden Behandlungsprozesse identifizieren.

5 Open-Source-Software Bausteine

Im Folgenden werden existierende Frameworks, die als eine Software-Basis bei der Entwicklung des Zielsystems eingesetzt werden, kurz beschrieben und evaluiert.

5.1 Workflow-Management mit jBoss jBPM

Im Bereich der Prozessunterstützung gibt es neben den kostenpflichtigen Produkten auch gute Open-Source-Projekte, die ebenfalls eine effiziente Prozessunterstützung ermöglichen. Eine Übersicht über Open-Source-Workflow-Engines kann man unter [Manageability] finden. Im Folgenden wird das jBoss jBPM Projekt betrachtet. jBPM erfreut sich einer ständig wachsenden Community im Internet und hat daher Chancen im Open-Source-Java-Bereich zum Standard zu werden. Die Roadmap des Projekts ist unter [jBPM09a] zu finden.

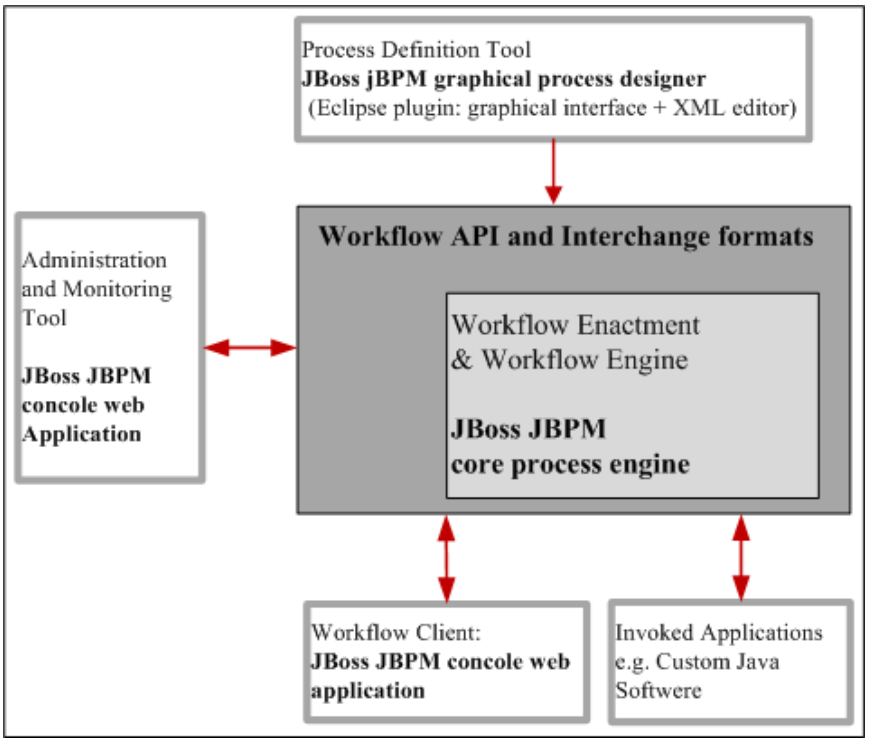


Abbildung 10: Architektur von jBoss jBPM (in Anlehnung an [Wohed07])

jBoss jBPM [jBPM09b] ist ein Workflow-Management-System, dessen Architektur in **Abbildung 9** skizziert ist. Die Architektur orientiert sich am Workflow-Referenzmodell der Workflow Management Coalition [WfMC09] und besteht aus den folgenden Hauptkomponenten:

- Die Workflow-Engine auch jBoss jBPM core component genannt, in der zur Laufzeit ein beschriebener Zustandsautomat abläuft.

- Das Prozess-Definiton-Tool auch JBoss jBPM Graphical Process Designer (GPD) genannt, ist ein Designer, welcher als Eclipse Plug-in zur Verfügung steht. Er bietet Unterstützung bei der Erstellung und Bearbeitung der Arbeitsabläufe in jPDL sowohl im graphischen und als auch im XML Format. jPDL (jBPM Process Definition Language) ist das von jBPM benötigte XML-Format.
- Die jBPM Console-Web-Application welche zwei Funktionalitäten bietet. Zum einen ist sie ein web-basierter Client, wodurch die Benutzer mit den Prozessinstanzen interagieren können. Zum anderen ist sie ein Tool zur Administration und zum Monitoring der laufenden Instanzen.

Eine Prozess-Definition in jPDL besteht aus mehreren Knoten und Transitionen zwischen diesen Knoten. Die graphischen Symbole der einzelnen Prozesskonstrukte und die graphische Repräsentation eines Beispiel-Modells sind **Abbildung 11** dargestellt. Die wichtigen Symbole sind:

- die Start- und End-Knoten, die den Anfang und das Ende des Prozesses explizit repräsentieren.
- Der State-Knoten zur Modellierung von Wartezuständen.
- Der „Node“ Knoten, der zur Einbettung vom eigenen Code dient. So kann man neues Verhalten definieren.
- Die Routing-Knoten wie „Fork“, „Join“ und „Decision“.

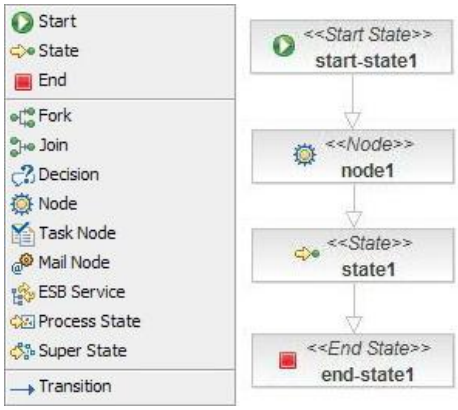


Abbildung 11: Die graphischen Symbole in jPDL und die graphische Repräsentation eines Beispiel-Modells

Die Geschäftslogik wird durch in Java-Code implementierten Aktionen repräsentiert. Die daraus gewonnenen Aktionen können dann verschiedenen Prozesskonstrukten zugewiesen werden, die bei entsprechenden Zustandsübergängen ausgeführt werden. So lassen die jPBM-Entwickler dem Anwender Flexibilität und Freiheit bei der Gestaltung des Verhaltens der Workflow-Anwendung.
Mit der Engine kann man einzelne Prozess-Instanzen eines Prozesses starten und deren Zustände und Transitionen überwachen. Eine Prozess-Instanz kann dabei beliebig viele

definierte Variable beinhalten. Eine detaillierte Beschreibung der Prozesskonstrukte und der Engine-Funktionalitäten sind unter [jBPM09c] zu finden.
jBPM ist auf einer einzigen Basis-Technologie aufgebaut: die Process-Virtual-Machine (PVM). Die PVM ist eine einfache Java-Bibliothek, die in allen Java-Umgebungen ausgeführt werden kann. Deswegen kann jBPM sowohl in einem stand-alone Programm oder Servlet-Kontainer als auch in jedem J2EE-konformen Applikationsserver betrieben werden. Im Falle der Notwendigkeit eines Persistenzmechanismuses, setzt jBPM auf Hibernate mit der Möglichkeit alle von ihm unterstützten relationalen Datenbanken zu benutzen. In **Abbildung 12** sind die erwähnten Bausteine von jPDL geschildert. Zusammenfassend lässt sich sagen, dass jBPM insgesamt die Möglichkeit bietet, in jeglicher Architektur und somit in vielen denkbaren Java-Anwendungen mit und ohne Applikationsserver eingesetzt zu werden.

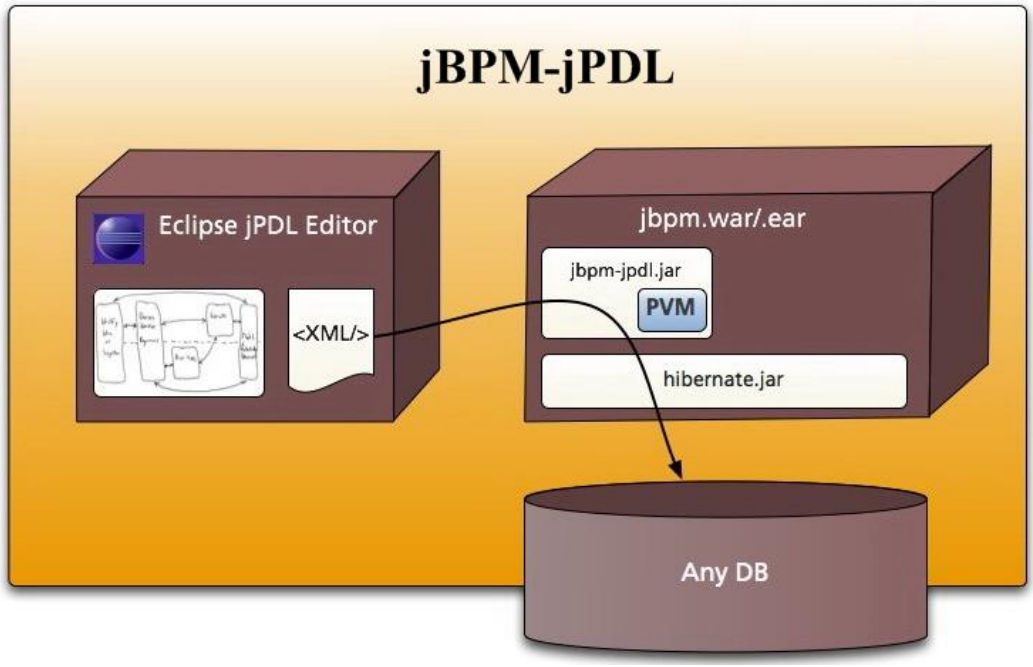


Abbildung 12: jBPM-jPDL Bausteine (Quelle [JBPM09b])

Indem die Grundlagen von jBoss jBPM kurz vorgestellt wurden, folgt im nächsten Abschnitt die Einführung eines Architekturstils, der als Basis für das Design von Web Services dient.

5.2 Der REST-Architekturstil

Diese werden meist mit SOAP oder XML-RPC in Verbindung gebracht. Mit dem REST-Architekturstil, repräsentiert im vorherigen Kapitel, können ebenfalls Web Services realisiert werden. Im Folgenden wird die REST und der Einsatz von der REST API beim Entwurf des Zielsystems beschreibt.

In seiner Dissertation [Fielding00] beschreibt R. T. Fielding einen Architekturstil für verteilte Hypermedia-Systeme, den er Representational State Transfer (REST) nennt. REST basiert auf Prinzipien, die in der größten verteilten Anwendung eingesetzt werden - dem World Wide Web. Das WWW stellt selbst eine gigantische REST Anwendung dar, welche deswegen als eine Schlüsselarchitektur für das WWW bezeichnet wird. Viele Web-Portale, Online-Shops oder andere Web-Anwendungen sind dabei ohne Absicht bereits als REST basierter Web Services verfügbar. Ressourcen im Internet wie Bilder, Web Seiten usw., können über URLs adressiert und angesprochen werden. Eine Web Anwendung repräsentiert eigentlich eine Ansammlung von Ressourcen. Man kann mittels HTTP Nachrichten an diese Ressourcen senden, beispielsweise durch den Aufruf von Seite in einem Browser wobei eine direkte Manipulation dieser Ressourcen nicht vorgesehen ist. Jeder Zugriff erfolgt indirekt über die der Resouce zugeordnete URI. Die Semantik des HTTP Protokolls wurde in REST übernommen. Eine zentrale Rolle dabei spielen die HTTP Methoden GET, PUT, POST und DELETE über welche die REST Ressourcen eine generische Schnittstelle besitzen.

- GET: Mit der Methode ruft der Client Daten vom Server auf
- POST: Mit der Methode können vorhandene Daten aktualisiert oder auf dem Server abgelegt werden
- PUT: Mit der Methode werden neue Daten/Ressourcen auf dem Server abgelegt
- DELETE: Mit der Methode kann der Client Daten auf dem Server löschen

Mit diesen Methoden können die meisten Anwendungsfälle abgedeckt werden im Gegensatz zu SOAP, in dem alle Methoden für jede Anwendung selbst definiert werden müssen. Der Begriff wird auch im weiteren Sinne verwendet, um grundsätzlich einfache Schnittstellen zu kennzeichnen, die Daten via HTTP übertragen, ohne etwa eine zusätzliche Transportschicht wie SOAP einzusetzen. Die folgenden Merkmale kennzeichen den REST Stil [Bayer02].

- Die Kommunikation erfolgt nach dem Pull-Prinzip. Der Client ist die aktive Seite, die vom passiven Server eine Repräsentation anfordert oder Ressourcen modifiziert.
- Ressourcen können über eine ihnen zugeordnete URI adressiert werden.
- Der Client kann Representationen von Ressourcen als Dokumente anfordern.
- Repräsentationen können auf Ressourcen verweisen. Diese können ihrerseits wieder Repräsentationen liefern.
- Es wird kein Clientstatus vom Server verfolgt. Jede Anfrage muss alle notwendigen Informationen zum Interpretieren der Anfrage beinhalten.

Für die Entwicklung REST-basierender Web Services existieren Frameworks für die unterschiedlichen Programmiersprachen wie „Django“ für Python, „Ruby on Rails“ für Ruby, „Restlet“ und „Jersey“ für Java usw. Da das Zielsystem komplett in Java entwickelt wird, folgt zunächst eine Vorstellung des Jersey-Projekts.

Jersey ist eine Open-Source Referenzimplementierung für JAX-RS (JSR 311): die Java-API für RESTful Webservices [JAXRS]. Jersey implementiert eine Unterstützung für die Notierungen definiert in JSR-311 und erleichtert die Erstellung von RESTful Webservices mit der Java-Programmiersprache und der Java-JVM für die Entwickler. Die Referenzimplementierung bietet noch zusätzliche Funktionen, die in der JSR spezifiziert sind. Die JAX-RS API zur Entwicklung von RESTful Webservices ist eine Java-API, die zur Vereinfachung der Entwicklung von Applikationen, die sich die REST-Architektur zum Nutzen machen, vorgesehen ist. Die API verwendet Java-Notierungen um diese Entwicklung zu vereinfachen. Die Entwickler versehen die Java-Klassen mit HTTP-spezifischen Notierungen mit dem Ziel Ressourcen und Aktionen, die auf diese Ressourcen anwendbar sind, zu definieren. Die folgenden Code-Zeilen zeigen ein einfaches Beispiel einer Root-Ressource-Klasse, die die JAX-RS Notierungen verwendet:

```
// Die Java-Klasse wird über den URI-Pfad "/helloworld"
ansprechbar sein
@Path("/helloworld")
public class HelloWorldResource {

       // Die Java-Methode verarbeitet HTTP GET Anfragen
       @GET
       // Die Java-Methode erzeugt MIME Media type "text/plain"
       Content
       @Produces("text/plain")
       public String getClichedMessage() {}

       // Die Java-Methode verarbeitet HTTP POST Anfragen
       @POST
       // Die Java-Methode erwartet MIME Media type "text/plain"
       Content
       @Consumes("text/plain")
       public void postClichedMessage(String message) {}

}
```

Die folgenden Notierungen sind im Beispiel verwendet:

- Der Wert der @Path-Notierung ist ein relativer URI-Pfad. Er zeigt über welchen URI-Pfad ist die Java-Methode anzusprechen.
- Die @GET-Notierung ist ein Anfrage-Methode-Bezeichner zusammen mit @POST, @PUT, @DELETE, @HEAD, die in JAX-RS definiert sind und den ähnlich benannten HTTP-Methoden entsprechen. Im konkreten Beispiel verarbeitet die Java-Methode HTTP GET Anfragen. Das Verhalten der Ressource ist also von der entsprechenden HTTP-Methode bestimmt
- Die @Produces-Notierung wird zur Spezifizierung der MIME Media Typen von Repräsentationen benutzt, die eine Ressource generieren und zum Client senden kann. Im Beispiel erzeugt die Java-Methode Repräsentationen, die der MIME Media Typ "text/plain" identifiziert
- Die @Consumes-Notierung wird zur Spezifizierung der MIME Media Typen von Repräsentationen benutzt, die eine vom Client empfangenen Ressource

verbrauchen kann. Im Beispiel erwartet die Java-Methode Repräsentationen vom MIME Media Typ "text/plain"

Die @Path-Notierung identifiziert die URI-Pfad Vorlage auf welche eine Ressource reagiert und wird auf der Klassenebene dieser Ressource spezifiziert. Der Wert dieser Notierung ist eine partielle URI-Pfad Vorlage, die der Basis-URI des Servers auf welchem die Ressource eingesetzt ist, relativ ist. Die Uri-Pfad Vorlagen sind URIs, die eingebetteten Variablen in der URI-Syntax haben können. Diese Variablen werden zur Laufzeit substituiert und mit geschweiften Klammern bezeichnet. Als Beispiel folgt eine mögliche @Path-Notierung mit eingebetteter Variable. Um den Wert dieser Variable zu beschaffen, kann man die @PathParam-Notierung als Methode-Parameter wie im Beispiel verwenden:

```
@Path("/users/{username}")
public class UserResource {
       @GET
       @Produces("text/xml")
       public String getUser(@PathParam("username") String userName) {}
}
```

Eine ausführliche Dokumentation bezüglich der Jersey-Funktionalitäten kann man unter [Jersey] finden. Im nächsten Kapitel werden die technisch-spezifischen Anforderungen analysiert und beschrieben, sowie eine Grobarchitektur des Systems erstellt.

6 Die Grobarchitektur des DMPS

Das Ziel verteilte Behandlungsprozesse informationstechnisch zu unterstützen, richtet sich in diesem Kapitel an die Erstellung einer Grobarchitektur des Zielsystems, die den ermittelten Anforderungen genügt. Ein institutionsübergreifender Prozess bedingt ein System, das geeignet ist in verteilten und heterogenen Umgebungen zu funktionieren und die dort vorhandene Infrastruktur zu nutzen. Sie sollen noch einfach zugänglich sein, so dass ihre APIs an die Verwendung bestimmter Software oder Betriebssysteme nicht gebunden sind. Im Folgenden wird die Grobarchitektur des Zielsystems präsentiert, die durch drei Schichten gekennzeichnet ist: eine Präsentationsschicht, eine Logikschicht und eine Datenschicht (vgl. **Abbildung 13**). So wird die Komplexität der Abhängigkeiten innerhalb des Systems reduziert und eine geringere Kopplung erzielt.

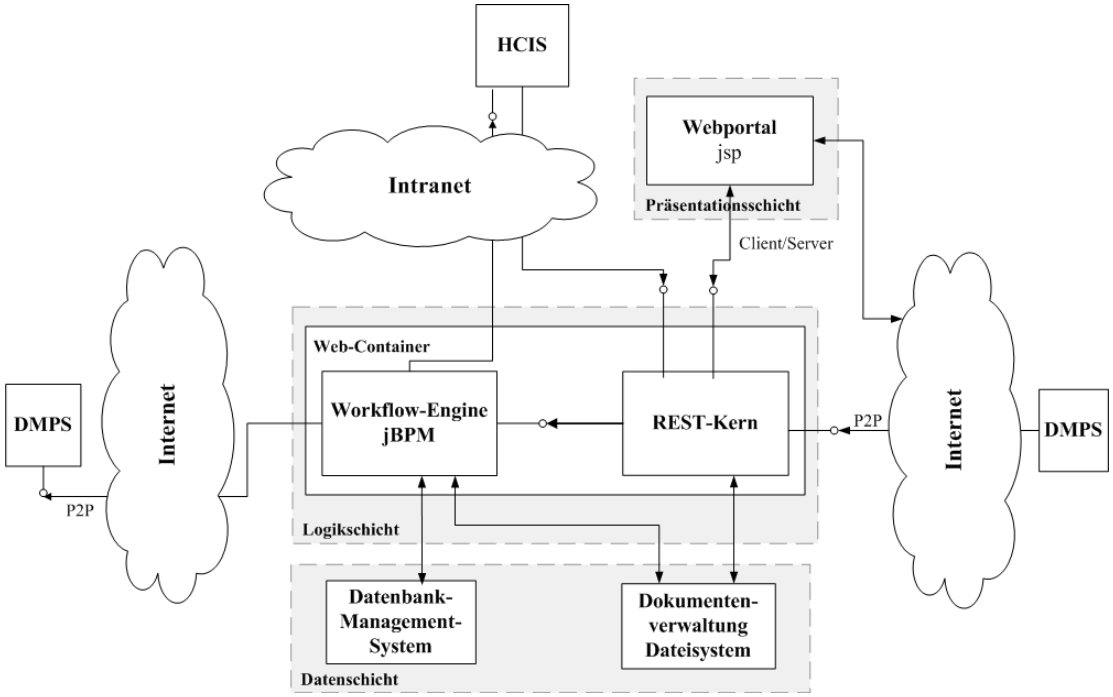


Abbildung 13: Die Grobarchitektur des Systems besteht aus drei Schichten: Präsentationsschicht, Logikschicht und Datenschicht

Die Logikschicht vereint die Anwendungslogik des Systems. Hier sind die jBPM Workflow-Engine und die REST Webservices als die Kernbausteine dieser Architektur zu finden. Die Kommunikation zwischen den Kernbausteinen soll über die jBPM API erfolgen. In der Workflow-Engine soll ein DMPS-Workflow die Hauptfunktionalität leisten. Dieser ist eine Sammlung von Funktionen, die das ganze Workflow-Modell für die Abwicklung von verteilten Behandlungsprozessen treiben. Die Initiierung eines solchen externen Prozesses soll im Workflow stattfinden indem dieser über die REST API eines externen HCIS mit integriertem DMPS Daten übergibt. Eine wesentliche Eigenschaft von jBPM ist, dass die verschiedenen Aspekte eines Ablaufes von der Implementierung der Anwendungslogik getrennt sind. Das hat zur Folge, dass die

Anwendungsstruktur leicht zu verstehen und der Kontrollfluss leicht nachvollziehbar und anpassbar ist.
Aus Sicht der eingeführten funktionalen Anforderungen, muss das DMPS sowohl die Kommunikation mit DMP-Systemen als auch mit non-DMPS-Benutzern ermöglichen. Die Außenschnittstelle soll als REST API entworfen werden und so die System-Funktionalitäten anderen DMP-Systemen bereitzustellen. Die Webservices können mit dem Jersey-Framework (vgl. Kapitel **5.2**) implementiert und als Servlets in eine Web-Container-Umgebung eingesetzt werden. Die Webservices sollen dann über einen ihnen zugeordneten URI-Pfad adressiert werden. Dieser URI-Pfad soll relativ zu einer DMPS-Basis-URI stehen, die bezüglich der eingesetzten Web-Container-Umgebung bei der Systemkonfiguration vorausgesetzt wird. Mit Rücksicht auf die DMP-Systeme, sind sie alle gleichberechtigt und können sowohl Dienste in Anspruch nehmen als auch Dienste zur Verfügung stellen. Deshalb stellen sie bei der Interaktion eine Peer-to-Peer (P2P) Verbindung her.
Den non-DMPS Benutzern können die Funktionalitäten über ein DMPS-Webportal angeboten werden. Das Webportal verkörpert die Präsentationsschicht des Systems und kann mit der Jsp-Technik implementiert werden. Beispielweise können niedergelassene Ärzte, die über kein integriertes DMPS verfügen, Dokumente über das DMPS-Webportal an das System übergeben. Das Zurücksenden der Resultate aus der Behandlung kann im Anschluss via SMTP erfolgen.
Weiterhin soll das DMPS in das lokale HCIS einfach integrierbar sein und ein unkomplizierter Informationsaustausch mit diesem ermöglichen. Eine REST-basierte Kommunikation bietet an dieser Stelle einige Vorteile. So spezifiziert REST einerseits uniforme und minimale Schnittstellen, die eine unkomplizierte Kommunikation bieten. Andererseits wird eine relativ einfache Integration in das HCIS ermöglicht, da über das HTTP kommuniziert wird und nicht über anwendungsspezifische Protokolle. Deshalb stellt das DMPS spezifische REST APIs zur Verfügung, über welche das lokale HCIS die Prozessabläufe in der Engine steuern kann. Die Rück-Kommunikation (DMPS-HCIS) hauptsächlich zwecks Notifikation bei Prozessänderungen und Prozessfehler oder bei der Übergabe der empfangenen Daten kann über generisch definierte REST APIs, die vom HCIS beliebig implementiert werden können. In dieser Hinsicht sollen die HCIS APIs über einen ihnen zugeordneten URI-Pfad adressiert werden, der relativ zu einer HCIS-Basis-URI steht. Diese Basis-URI wird ebenfalls bei der Konfiguration des DMPS vorausgesetzt.
Die Datenhaltungsschicht ist ein untrennbarer Teil jeder Software-Architektur. Ein Teil dieser Schicht wird vom Datenbank-Management-System repräsentiert. Das DMS soll mit der Workflow-Engine in Interaktion stehen und die Persistenzhaltung der Prozessdefinitionen und der Zustände dieser Engine leisten. Die Persistierung mit Hilfe des DBMS soll der gestellten Anforderung an Zuverlässigkeit des Systems genügen. Diese Schicht soll zudem noch einen Mechanismus zur Dokumentenverwaltung bereitstellen. Da die Implementierung eines Dokumentenmanagementsystems nicht Teil dieser Arbeit ist, kann bei der Entwicklung das Dateisystem zur Verwaltung der Dokumente verwendet werden. Eine geeignete Verzeichnisstruktur in dem Dateisystem soll zur Speicherung der prozessbegleitenden Dokumente dienen.

7 Der DMPS-Systementwurf

Der Systementwurf gehört zu den Hauptaktivitäten der Entwicklung des DMPS. Nach erfolgter Problemidentifikation und Anforderungsdefinition wird hierbei die Grundlage für die Software-Implementierung im Kapitel 8 geschaffen. Dieser Teil der Arbeit umfasst insbesondere die Identifikation und Spezifikation von einzelnen Bausteinen der Architektur (z.B. Komponenten, Schnittstellen, usw.) des zu entwickelnden Systems und beginnt mit der Spezifikation des DMPS Prozess-Modells.

7.1 Das DMPS Prozess-Model

Auf Basis des diagnostisch-therapeutischen Vorgangs, welcher in Kapitel 3 vorgestellt wurde, wird hier der Ansatz zur IT-Unterstützung von verteilten diagnostisch-therapeutischen Behandlungen beschrieben, präzisiert und anschließend durch Einsatz von Zustandsdiagramme modelliert. An dieser Stelle ist wichtig zu erwähnen, dass dieses Vorgehen noch keine spezifische Basissoftware impliziert und deswegen spielt die geplante Verwendung eines Workflow-Management-Systems noch keine Rolle. Erst in der Implementierungsphase, soll von der Beschreibung der Arbeitsabläufe im Anwendungsbereich zu einer Spezifikation von Workflows übergegangen werden.
Abbildung 4 aus Kapitel 3 zeigt den generellen Ablauf einer diagnostisch-therapeutischen Behandlung. Dabei wird nicht unterschieden, ob der Patient während der Behandlung eine oder mehrere medizinische Organisationseinheiten durchläuft. Deshalb kann man die diagnostischen und therapeutischen Maßnahmen auf dem Behandlungspfad des Patienten während einer Behandlung als lokale oder externe, bezüglich der Organisationseinheit in der sie durchgeführt werden, abgrenzen. Dies ist aus Sicht des DMPS eine wichtige Unterscheidung, da die Workflow-Anwendung den Austausch von Dokumenten im Rahmen eines verteilten Behandlungsprozesses unterstützen soll. Es wird während der Modell-Erstellung nur die Abwicklung von externen Behandlungen in Betracht gezogen. Diesbezüglich steht die Steuerung der verteilten Behandlung aus Sicht der Interaktion mit den externen Systemen im Vordergrund und nicht die konkrete Art der Behandlung.
Im Rahmen dieses Abschnittes wird aus dem klassischen diagnostisch-therapeutischen Zyklus ein erweitertes Modell abgeleitet und als Zustandsdiagramm modelliert, welches die Grundlagen für das Workflow-Modell in der Implementierungsphase bildet. Dieser Zyklus wird als erstes ganz grob aus dem Blickwinkel der Ausführung des diagnostisch-therapeutischen Vorgangs betrachtet und präzisiert.
Infolge der eingeführten Abgrenzung nach lokalen oder externen Maßnahmen werden zwei wichtige Zustände definiert: „decision making“ und „extern treatment running“. Der erste Zustand soll den ganzen diagnostisch-therapeutischen Zyklus hinsichtlich der lokalen Behandlungsabläufe im Rahmen einer medizinischen Organisationseinheit umfassen. Zum anderen bildet der zweite „extern treatment running“ Zustand alle medizinischen Abläufe ab, die als Teil eines zentral betrachteten Behandlungsprozesses außerhalb einer medizinischen Organisationseinheit ablaufen und kann als eine Erweiterung des klassischen Zyklus betrachtet werden. Diese zwei eingeführten

Zustände sowie die folgenden Konstrukte in **Abbildung 14** bilden das erweiterte Modell zur Unterstützung von verteilten Behandlungsprozessen. Es ist zu merken, dass dieses Zustandsdiagramm teilweise auf dem klassischen diagnostisch-therapeutischen Zyklus aufbaut und diesen um die Kommunikation mit externen Institutionen ergänzt.
Als erstes ist es wichtig zu erklären, mit wem kommuniziert wird und wie das Modell dabei reagiert. Im Zustandsdiagramm (vgl. **Abbildung 14**) werden die Interaktionen mit den unterschiedlichen Systemen unterschiedlich gefärbt. Zum einen interagiert das DMPS mit dem bestehenden lokalen HCIS über die lokale APIs. Zum anderen erfolgt eine Kommunikation mit externen DMP-Systemen. Neben dieser Unterscheidung sind noch die Prozess-Initiatoren abgegrenzt.

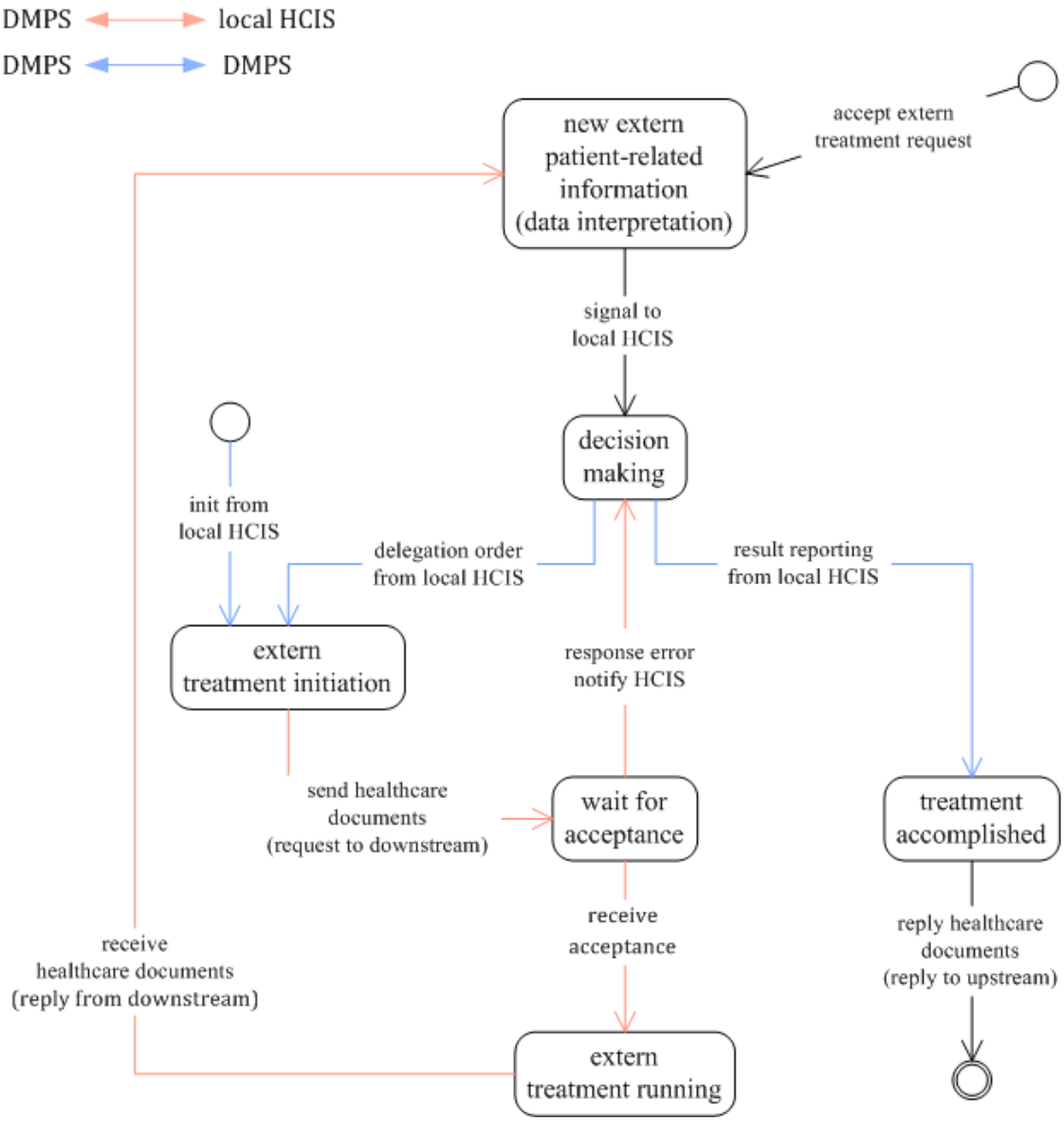


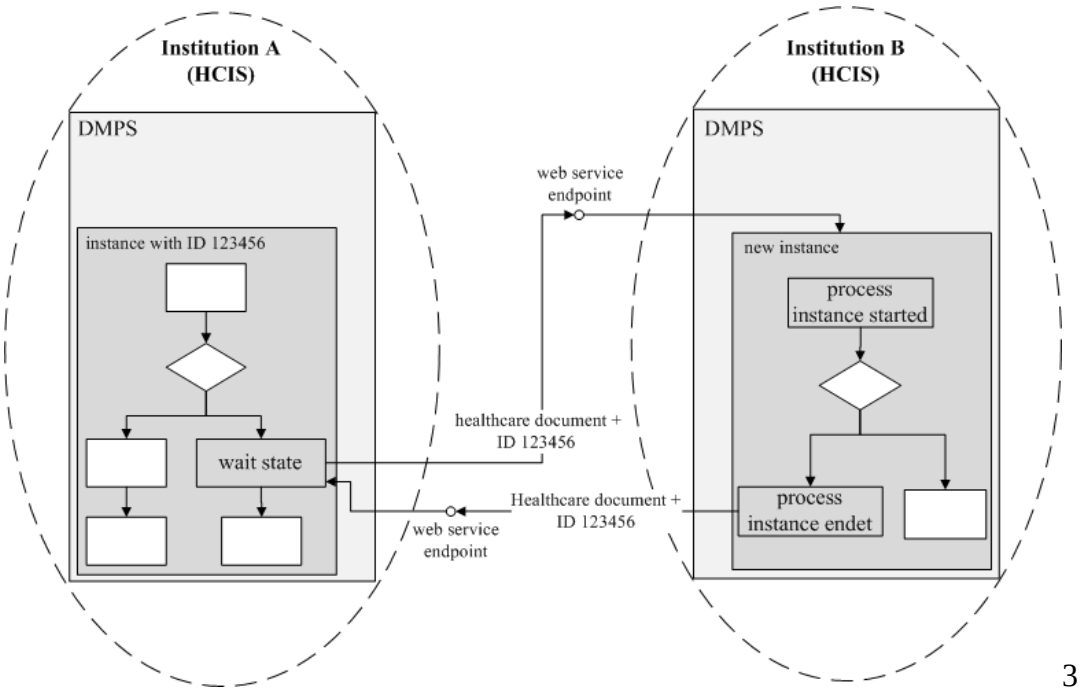
Abbildung 14: Das DMPS Prozess-Model zur Unterstützung von verteilten Behandlungsprozessen als Zustandsdiagramm

In Anlehnung an das Diagramm werden zwei Wege zur Initiierung von verteilten diagnostisch-therapeutischen Prozessen vorgesehen. Einerseits triggert das lokale HCIS in dem lokalen DMPS die Initiierung einer externen Behandlung. Dies geschieht durch ein Signal vom HCIS an dem DMPS indem der Prozess im „extern treatment initiation“ Zustand überführt wird (vgl. **Abbildung 14**). In diesem Zustand erfolgt das Versenden der patient-bezogenen medizinischen Dokumente an die externe Institution. Diese Aktion kann man auch als „request to downstream“ bezeichnen. Im Fehlerfall des Versendens wird das HCIS über den Fehler informiert und der Prozess im „decision making“ überführt. Im Erfolgsfall (die externe Behandlung wird akzeptiert) bleibt der Prozess im „extern treatment running“-Wartezustand während des Verlaufs der Behandlung. Nach der Beendigung schickt die externe Institution die patient-bezogenen Informationen zurück (reply from downstream). Es wird erkannt, zu welchem laufenden Prozess die empfangenen Informationen gehören und das lokale HCIS benachrichtigt, wobei der Prozess wieder im „decision making“ Zustand versetzt wird. In diesem Fall handelt das lokale HCIS als „upstream participant“ gegenüber der Abwicklung der Behandlung.
Andererseits wird durch den Empfang von Dokumenten über die DMPS-API eine lokale Behandlung (externe Behandlung aus Sicht der anfordernden Institution) initiiert. In diesem Fall handelt die Institution als „downstream participant“. Dabei wird das lokale HCIS über seine API benachrichtigt und der DMPS-Prozess im „decision making“ überführt. An dieser Stelle erfolgt in der Organisationseinheit ein lokaler Behandlungsprozess und der Prozess bleibt solange im Wartezustand, bis ein Befehl vom HCIS über die lokale API bezüglich weiterer Behandlungsmaßnahmen ankommt. Es können entweder weitere externe Behandlungen eingeleitet oder die Behandlung beendet und die Resultate aus der Behandlung dem externen DMPS verkündet werden. Dieses Modell bildet die Grundlage bei der Entwicklung der Workflow-Management-Anwendung. Nachdem die jBPM-Workflow-Engine als Kernbaustein der Anwendung vorgestellt wurde, folgt im Kapitel 8 die konkrete Überführung des Prozessmodells in ein Workflow-Modell bezüglich der konkreten Workflow-Sprache sowie die Workflow-Implementierung.

7.2 Die prozessbegleitenden Informationen

jBPM verwendet Hibernate als Persistenzframework zur Persistierung von Zuständen und Kontextinformationen während der Prozessausführung. Wenn die Engine eine neue Prozessinstanz startet, generiert jBPM eine Instanz-ID. Diese wird der Prozessinstanz in der Datenbank als eindeutige Identifikation zugewiesen. Mit der Kenntnis dieser ID ist man in der Lage über die von jBPM angebotenen API, die richtige Prozessinstanz zu identifizieren und zu steuern (zum Beispiel einen Prozess-Statuswechsel triggern, Prozessinstanz-Variablen anlegen oder Prozessinstanzen beenden). Bei einer DMPS-DMPS-Interaktion, sollen medizinische Dokumente zwischen Prozessinstanzen in diesen Systemen ausgetauscht werden, die in unterschiedlichen und autonomen Workflow-Engines laufen. In dieser Richtung ergeben sich Fragen zur Identifizierung

der einzelnen DMP-Systeme und der Prozessinstanzen in diesen Systemen. Das nächste Beispielszenarium (vgl. **Abbildung 15**) schildert diese Problematik genauer. Zwei Healthcare-Institutionen, die über das DMPS verfügen, partizipieren an einem verteilten Behandlungsprozess. Darüber hinaus laufen in den autonomen Workflow-Engines zwei Prozessinstanzen, die medizinischen Daten austauschen sollen.
Die Prozessinstanz mit ID 123456 in HCIS-A leitet eine externe Behandlung ein, indem sie Dokumente an HCIS-B über die verfügbaren Webservices übergibt. Die Prozessinstanz in HCIS-A soll also nur den Webservice-Endpunkt von HCIS B kennen. Beim Empfang der Dokumente von HCIS-A, startet die Workflow-Engine in HCIS-B eine neue Prozessinstanz und initiiert so eine Behandlung. Inzwischen bleibt die Prozessinstanz in HCIS-A in einem Wartezustand. Die Prozessinstanz in HCIS-B wird beendet und nun sollen die Behandlungsresultate zurück an die Prozessinstanz in HCIS-A geschickt werden. An dieser Stelle rücken die beschriebenen Probleme in den Vordergrund: wie soll die Prozessinstanz in HCIS-B das HCIS-A adressieren und wie identifiziert die Workflow-Engine in HCIS-A die richtige Prozessinstanz zu welcher die empfangenen Dokumente gehören (in diesem Fall mit ID 123456)?



3

Abbildung 15: Ein Beispielszenarium mit zwei HCI-Systemen die medizinischen Daten austauschen. Zusätzlich müssen die Instanz-ID und der Webservice-Endpunkt prozessbegleitend übermittelt werden.

Da ein HCIS über die vom DMPS angebotenen RESTful Webservices identifiziert werden kann, stellt sich als erstes die Anforderung zur prozessbegleitenden Übermittlung der Rückantwort-Webservice-Endpoints. Diese Art von Information würde das Problem bei der Identifizierung des für den Empfang einer Rückantwort nach der Beendigung einer externen Behandlung zuständigen DMPS lösen. Es wurde schon

angedeutet, dass man über die Instanz-ID die zugehörige Prozess-Instanz im Rahmen einer Workflow-Engine (DMPS) eindeutig bestimmen kann. Diese Tatsache würde das Problem bei der Prozessinstanz-Wiederfindung lösen, wenn die Instanz-IDs ebenfalls prozessbegleitend übermittelt werden.
Verteilte Behandlungsprozesse können sich über mehrere Institutionen erstrecken. Eine komplette Behandlung in diesem Fall kettenförmig oder sternförmig über Institutionen verteilt sein (vgl. **Abbildung 16**). Es ist dabei wichtig für die Prozessteilnehmer, dass sie den Überblick über die Behandlungsschritte nicht verlieren und jederzeit den bisherigen Behandlungsablauf zurückverfolgen können. Man könnte also durch eine entsprechende prozessbegleitende Protokollierung, eine Behandlungshistorie schaffen und bereitstellen, über welche die teilgenommenen Institutionen während der externen Behandlungsmaßnahmen eindeutig nachvollzogen werden können. Nachdem die prozessbegleitenden Informationen identifiziert wurden, werden im Folgenden zwei Ansätze evaluiert und eine Möglichkeit zur Übermittlungen dieser Informationen erarbeitet.
Da CDA-Dokumenten prozessbegleitende Informationen zum Arbeitsablauf fehlen, wäre es einerseits möglich diese Informationen mit den einzelnen CDA-Dokumenten zu übertragen. Dazu könnte der CDA-Header aller an der Prozessinstanz beteiligten Dokumente um Prozessinformation erweitert werden.

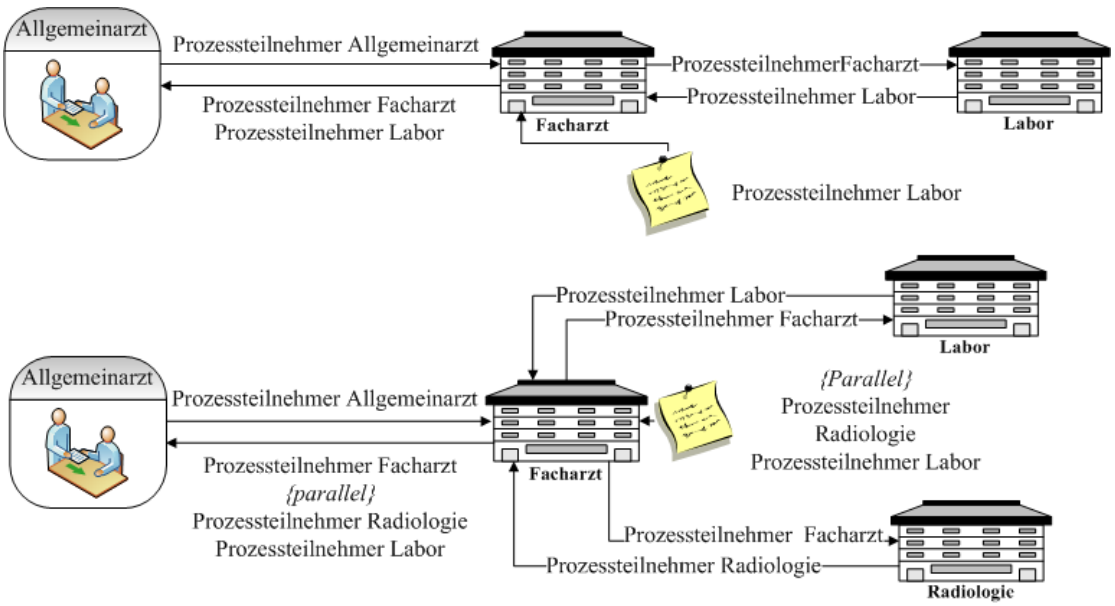


Abbildung 16: Die Darstellung zeigt zwei Szenarien mit einer kettenförmigen und einer sternförmigen Struktur des verteilten Behandlungsprozesses und die möglichen Behandlungshistorien, die zwischen den Teilnehmer ausgetauscht werden können.

Damit würde ein Prozess aus seinen CDA-Dokumenten die Information automatisch gewinnen, sofern die Systeme die Information verstehen. Andere Systeme, die die Information nicht verstehen, können das CDA-Dokument normal bearbeiten und die Information ignorieren. Dieser Ansatz ist also kompatibel zur CDA-Philosophie. Die Vorteile dabei sind, dass nur CDA-Dokumente zwischen den einzelnen Teilnehmer

übertragen werden. Die Vorgehensweise hat auch Nachteile. Zum einen werden häufig mehrere Dokumente bezüglich einer externen Behandlung gleichzeitig übermittelt. Dies erfordert, dass in jedem einzelnen CDA-Dokument die gleiche Information hinzugefügt wird. Dabei werden aus Sicht der System-Performance mehrere unerwünschte Schreib-/Lese-Operationen sowie XML-Parsing-Operationen ausgeführt. Zudem weist diese Vorgehensweise Redundanzen bezüglich der gespeicherten Prozessinformation auf. Zum anderen müssen die Behandlungshistorien in jedem Behandlungsknoten explizit zentral verwaltet und zwischen den einzelnen Dokumenten synchronisiert werden. Dies wird besonderes ersichtlich, wenn Behandlungen parallel ablaufen.
Betrachten wir noch einmal das zweite Szenarium in **Abbildung 16**. Nach der lokalen Behandlung beim Allgemeinarzt wird eine externe Behandlung beim Facharzt eingeleitet wobei der Allgemeinarzt als Teilnehmer in der Behandlungshistorie auch als Information einfließt. Vom Facharzt werden als Folge zwei externe parallele Labor- und Radiologie-Untersuchungen initiiert. Beim Labor und bei der Radiologie fließen der Allgemeinarzt und der Facharzt sequenziell als Behandlungshistorie ein. Beim Rückfluss der Resultate zum Facharzt, werden einerseits die Historie aus dem Behandlungspfad Allgemeinarzt -> Facharzt -> Labor und andererseits die Historie aus dem Behandlungspfad Allgemeinarzt -> Facharzt -> Radiologie in den CDA-Headern gespeichert. Da nach Beendigung der Behandlung beim Facharzt, die Informationen zurück an dem Allgemeinarzt übergeben werden müssen, hat hier das DMPS die Aufgabe die Behandlungshistorien aus den einzelnen Dokumenten zentral zu verwalten, zu synchronisieren und zu konsolidieren, so dass beim Allgemeinarzt ersichtlich wird, dass parallelen Behandlungen im Labor und Radiologie stattgefunden haben.
Als Herleitung aus der obigen Betrachtung zur Übermittlung von Prozessinformationen könnte alternativ eine Art „Prozesszustandsdokument“ in Form eines XML-Dokuments als Bezugsobjekt der Prozessinstanz instrumentiert werden. Dabei ist zusätzlich zu den einzelnen Dokumenten in einem Prozess ein übergeordnetes Dokument zu übertragen, zu pflegen und aktuell zu halten, mit dem die beteiligten Systeme auch umgehen können müssen. Abgesehen davon wird nur ein Dokument gepflegt und keine redundanten Informationen übermittelt. Dazu werden die Behandlungshistorien einfach in einem Dokument verwaltet und aktualisiert und nicht aus den einzelnen CDA-Dokumente wiederholt ausgelesen und von jedem DMPS separat verwaltet. Abgesehen von der Notwendigkeit ein zusätzliches Dokument zu übertragen, bietet dieser Ansatz im Vergleich zu der Header-Erweiterung mehrere Vorteile und wird in Bezug auf die Ziel-Anwendung verwendet:

- keine CDA-Header-Erweiterung. Die CDA-Dokumente werden vom DMPS einfach nur übertragen.
- keine redundanten Informationen in den CDA-Headern.
- es finden wenige Schreib-/Lese- und XML Parsing-Operationen statt, da nur auf ein Dokument zugegriffen wird (bessere Performance).
- Einfachheit und Übersichtlichkeit bei der Verwaltung der Behandlungshistorien. Die DMP-Systeme müssen nur ein prozessbegleitendes Dokument aktualisieren

und nicht die Historien aus den einzelnen CDA-Dokumenten auslesen und in jedem DMPS zentral verwalten.

Vor der Beschreibung der XML Struktur des Prozesszustandsdokuments, wird zuerst die Terminologie eingeführt, welche bei der Definition dieser Struktur verwendet wurde. **Abbildung 17** schildert das XSD-Schema.

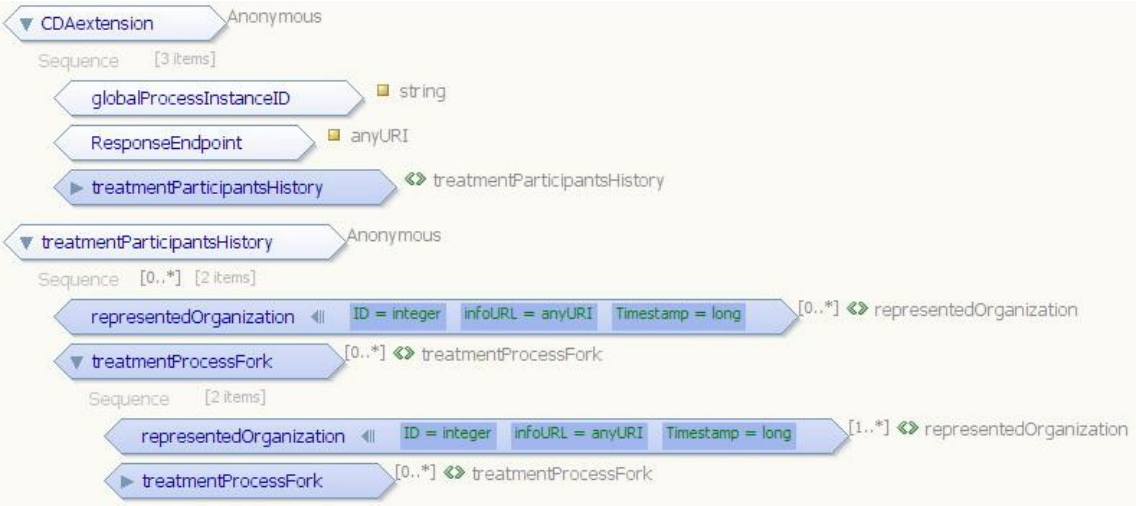


Abbildung 17: Das XSD-Schema des Prozesszustandsdokumentes

Mit dem „globalProcessInstanceID“-Element werden die zuvor in diesem Kapitel beschriebenen Prozessinstanz-IDs im Dokument gespeichert und prozessbegleitend übertragen. Das „ResponseEndpoint“-Element definiert den Teil der XML-Struktur, die sich auf die Rückantwort-Webservice-Endpoints der DMP-Systeme bezieht. Interessant ist der „treatmentParticipantsHistory“-Teil dieser Struktur, der als Protokoll-Struktur zur Veranschaulichung der Behandlungshistorie dient. Diese Teil-Struktur besteht aus zwei Schema-Elementen und ist rekursiv aufgebaut, so dass jeder beliebig-verteilte Behandlungsablauf beschrieben werden kann. Eine im Prozess teilnehmende Institution wird mit dem „*representedOrganization“*-Element dargestellt. Es hat drei Attribute von denen zwei optional sind:

```
<xs:element name="representedOrganization">
        <xs:complexType>
            <xs:attribute name="ID" type="xs:integer" use="required"/>
            <xs:attribute name="infoURL" type="xs:anyURI" use="optional"/>
            <xs:attribute name="Timestamp" type="xs:long" use="optional"/>
        </xs:complexType>
</xs:element>
```

Das ID-Attribut dient zur eindeutigen Identifikation des einzelnen Elements innerhalb eines Dokuments bei der Synchronisation von Historien und wird von der internen DMPS-Logik gebraucht. Die eigentlichen Methoden zur Vergabe dieser ID’s und zur Synchronisation werden im nächsten Kapitel näher erläutert. Das infoURL-Attribut ist eine Referenz zu der Info-Seite der teilnehmenden Healthcare-Organisation. Das letzte Element-Attribut ist der Zeitstempel, der vor der Übertragung der Dokumente bei der Initiierung einer externen Behandlung generiert wird. Mit dieser Information kann jedes

DMPS die zeitliche Reihenfolge und den Dauer der Behandlungen genau bestimmen und als Status-Information dem HCIS bereitstellen.
Die „treatmentParticipantsHistory“-Struktur beginnt mit einer Sequenz von null bis unendlich vielen Organisationen (vgl. **Abbildung 17**). Eine solche sequenzielle Folge von Organisationen liegt im Fall eines verteilten kettenförmigen Behandlungsprozesses vor. Wenn aber ein Szenarium vorliegt, bei dem zwei oder mehreren parallelen Behandlungen initiiert werden, rückt die „treatmentProcessFork“-Struktur in den Vordergrund. Sie kennzeichnet eine Prozessverzweigung beim Behandlungsablauf, d.h. es wurden parallele Behandlungen durchgeführt. Der Prozess der Generierung einer Historie wird mit dem folgenden Beispiel veranschaulicht (vgl. **Abbildung 18**).

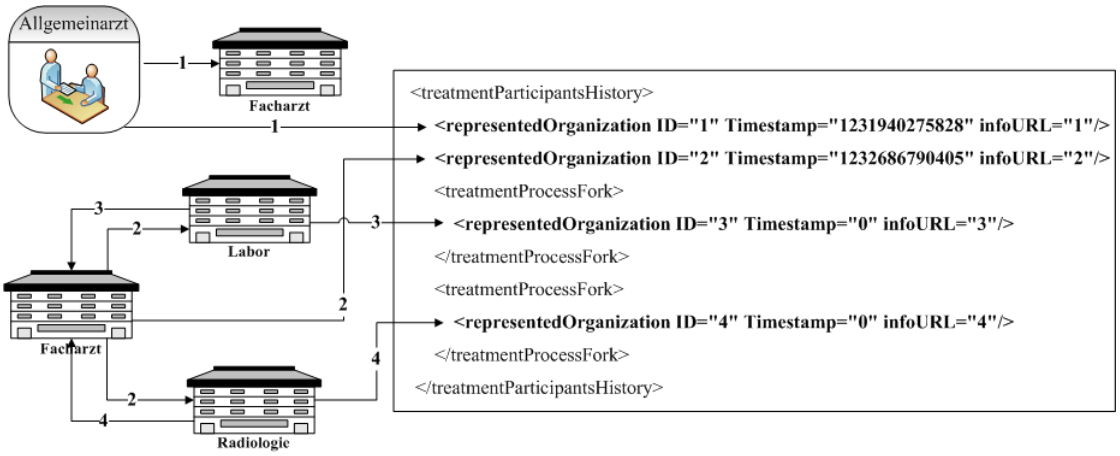


Abbildung 18: Die Historiengenerierung am Beispiel eines verteilten Behandlungsprozesses mit zwei parallelen Behandlungen

Das XSD-Schema sowie zwei Prozessdokumente als Beispiel nach einer sequenziellen und parallelen Behandlung sind im Anhang **A.1** zu finden.

7.3 Die Datenschicht des Systems

Die Datenhaltungsschicht ist ein unverzichtbarer Teil jeder Software-Architektur. Neben der relationalen Datenbank zur Persistierung von Prozessdefinitionen und Zustände in jBPM, müssen die für jeden verteilten Behandlungsprozess begleitenden Dokumente verwaltet werden. Da die Implementierung eines DMS nicht Teil dieser Arbeit ist, wird hier zur Verwaltung der Dokumente das Dateisystem verwendet. In diesem Abschnitt wird eine Verzeichnisstruktur vorgestellt, die zur Verwaltung der prozessbegleitenden Dokumente dient. Die Struktur ist hierarchisch aufgebaut und wird als ein Verzeichnisbaum in **Abbildung 19** dargestellt.
Das DMPRepos-Verzeichnis ist das Wurzelverzeichnis der Dateistruktur. Auf der zweiten Ebene der Struktur werden die Arbeitsverzeichnisse für die einzelnen Prozessinstanzen und ein Input-Verzeichnis für die temporäre Speicherung der empfangenen Dokumente angelegt. Wenn eine Prozessinstanz gestartet wird, legt das DMPS ein Verzeichnis mit der Instanz-ID als Name an. Da die IDs eindeutig generiert

werden, besteht keine Kollisionsgefahr beim Anlegen der Verzeichnisse. Jeder Instanz-Ordner enthält drei Unterordner. Der „reqtodownstream_docstore“- Ordner (request to downstream) enthält die CDA-Dokumente, die bei Forderung externer Behandlungen gesendet werden. Die empfangenen Dokumente nach der Beendigung der externen Behandlungen werden in dem „repfromdownstream_docstore“ (reply from downstream) abgelegt. Diese Dokumente werden häufig von mehr als ein DMPS zurück gesendet. Um diese zu unterscheiden, werden beim Empfang der Dokumente Unterordner mit Namen als Zeitstempel in Millisekunden generiert und angelegt. Durch diese Zuweisung von Namen, wird sowohl die Eindeutigkeit sichergestellt als auch die genaue Empfang-Zeit notiert und als Information dem DMPS bereitgestellt. Der letzte „reptoupstream_docstore“-Ordner (reply to upstream) soll die CDA-Dokumente enthalten, die als Rückmeldung bei der Beendigung einer lokalen Behandlung an den Auftraggeber zu senden sind. Die Arbeitsverzeichnisse der einzelnen Prozessinstanzen werden nach dem Ende deren Ausführung aufgeräumt.

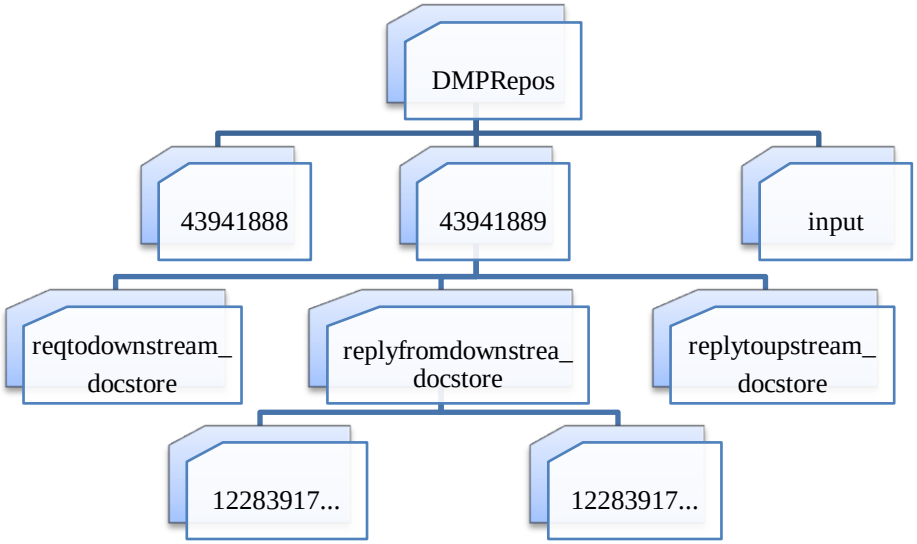


Abbildung 19: Die DMPS-Verzeichnisstruktur zur Verwaltung der ausgetauschten Dokumente

7.4 Der DMPS REST-Kern

Dieses Kapitel beschäftigt sich mit der Möglichkeit des Systems nach außen zu kommunizieren. Die Schnittstellen zum Anwender haben die Aufgabe, die Interaktion grundsätzlich zu ermöglichen und gleichzeitig die Menge der möglichen Interaktionen und Rückmeldungen auf ein hinreichendes und überschaubares Maß zu reduzieren. Die Benutzungsschnittstellen sind in diesem Kapitel in zwei Bereiche aufgeteilt: die DMPS API und die HCIS API.

7.4.1 Die DMPS API

Die Funktionalitäten des DPMS werden über die REST-API anderen Systemen zur Verfügung gestellt. Die Schnittstellen werden grundsätzlich in zwei Bereiche, bezüglich

der Systeme mit denen die Interaktion erfolgt, aufgeteilt. Einerseits bietet DMPS den externen HCIS den Austausch von Dokumenten und so die Initiierung von Behandlungen über die REST API. Andererseits interagiert das System über die APIs mit dem lokalen HCIS (vgl. **Abbildung 20**).

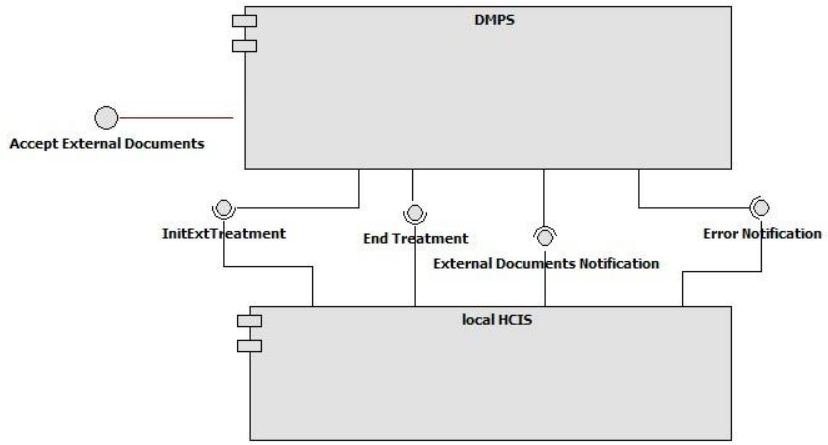


Abbildung 20: Das DMPS kommuniziert über APIs mit externen Anwendern sowie mit dem lokalen HCIS

Die Hauptfunktionalität wird von der „Accept External Documents"-Schnittstelle erbracht. Sie baut auf die POST Methode des HTTP-Protokolls auf und erwartet die HTTP POST-Anfragen über den */reqrep* URI-Pfad. Der Name des Pfads steht für Request/Reply, da nicht nur neue Behandlungsanfragen sondern auch Rückmeldungen von bereits initiierten externen Behandlungen über die API zu empfangen sind. Zum Versand von mehreren CDA-Dokumenten sowie der Prozesszustandsdokumente, wird die Anfrage als MIME-Multipart-Type aufgebaut. Eine Multipart-Message enthält mehrere Bodyparts, die durch eine Grenzlinie abgegrenzt werden. Dies erlaubt das gleichzeitige Versenden von mehreren Dokumenten in einer Nachricht. Die Logik hinter der Schnittstelle kann man in sechs Schritten abgrenzen.

Schritt 1: Die empfangenen Nachrichten werden auf Gültigkeit überprüft und die einzelnen Dokumente aus der Multipart-Nachricht gewonnen.

Schritt 2: Die Logik iteriert durch die empfangenen Dokumente und sucht das Prozesszustandsdokument. Die Suche erfolgt durch die Validierung der einzelnen Dokumente mit dem XSD-Schema des Prozesszustandsdokuments. Wenn kein valides Prozesszustandsdokument gefunden wird, liegt der Fall, wenn ein non-DMPS mit dem System kommuniziert vor.

Schritt 3: Es werden den Rest der Dokumente mit dem HL7 CDA XSD-Schema validiert. Wenn keine valide CDA-Dokumente gefunden werden, empfängt der Sender als Antwort eine Fehlermeldung.

Schritt 4: Die Logik initialisiert eine neue Prozessinstanz über die jBPM API. Dieser Instanz wird eine eindeutige Instanz-ID von der Workflow-Engine zugewiesen. Mit der daraus gewonnenen ID wird die File-System-Struktur für die Prozessinstanz aus Kapitel 7.2 generiert und die empfangenen Dokumente in das jeweilige Verzeichnis kopiert.

Schritt 5: Die Prozessinstanz wird über die jBPM API gestartet.

Schritt 6: Eine Antwort mit Status Code „201 Created“ und die URI als Referenz zu der Ressource wird an den Sender geschickt.

Die Interaktion mit dem lokalen HCIS erfolgt ebenfalls über die REST-APIs. Die bereitgestellten Funktionalitäten dieser APIs finden ihre Anwendung auf die einzelnen Prozessinstanzen. Es wurde bereits im Kapitel 7.2 festgestellt, dass die Identifikation und die Steuerung einer Instanz die Kenntnis ihrer eindeutigen ID bedarf. Diese Feststellung setzt voraus, dass diese APIs auf Kenntnis der IDs beruhen. Darüber hinaus muss das HCIS vom DMPS über die neu generierte ID benachrichtigt werden, wenn eine neue Prozessausführung vorliegt.

Zum einen bietet eine REST-API die Initiierung externer Behandlungen, d.h. CDA-Dokumente und Prozesszustandsdokument an einem externen Endpoint zu schicken. Hier unterscheidet man zwei mögliche Fälle, die über zwei URI-Pfade verfügbar sind. Im ersten Fall kann man über den */initiateExtTreatment* URI-Pfad eine neue Prozessinstanz starten und im Anschluss eine externe Behandlung initiieren. Die Schnittstelle ist auf der POST Methode des HTTP-Protokolls aufgebaut, da das HCIS beim Aufruf dieser Methode auch die zu versendenden CDA-Dokumente an das DMPS übergibt. Die Logik initialisiert dann eine neue Prozessinstanz über die jBPM API, generiert die File System Struktur, setzt die Prozessinstanz im „extern treatment initiation“-Zustand und startet diese. Die übergebenen CDA-Dokumente sowie das Prozesszustandsdokument werden dann von dem Prozess versendet. Im Erfolgsfall empfängt das HCIS eine HTTP-Anwort mit Status Code „200 OK“ und mit der ID der gestarteten Prozessinstanz. Anderenfalls empfängt das HCIS eine Fehlermeldung mit Status Code „409 Conflict“.

Im zweiten Fall wird bei einer bereits laufenden Prozessinstanz, d.h. die Instanz wurde bereits als Teil eines verteilten Behandlungsprozesses initialisiert und gestartet, über den */{instanceid}/initiateExtTreatment* URI-Pfad eine externe Behandlung initiiert. Die Schnittstelle ist ebenfalls auf der POST Methode des HTTP-Protokolls aufgebaut, da das HCIS beim Aufruf dieser Methode auch die zu versendenden CDA-Dokumente an das DMPS übergibt. Die Logik überprüft dann über die jBPM-API den momentanen Ausführungszustand der Prozessinstanz. Eine externe Behandlung darf nur dann initiiert werden, wenn sich die Instanz im Zustand „decision making“ befindet. Wenn der momentane Zustand tatsächlich „decision making“ ist, erfolgt ein Zustandsübergang vom „decision making“ zu „extern treatment initiation“. Der Prozess macht dann den Rest und schickt im Erfolgsfall eine HTTP-Anwort mit Status Code „200 OK“ an das HCIS. Anderenfalls empfängt das HCIS eine Fehlermeldung mit Status Code „409 Conflict“.

Zum anderen bietet das DMPS die Möglichkeit laufende Prozessinstanzen und das Warten auf die initiierten externen Behandlungen über die */{instanceid}/endTreatment* und */{instanceid}/escapeExtTreatment* URI-Pfade zu beenden. Die Methoden bauen auf der GET Methode des HTTP-Protokolls auf und sind durch die „EndTreatment“-Schnittstelle in **Abbildung 20** repräsentiert.

Bei der ersten Methode wird über die jBPM-API als erstes der momentane Ausführungszustand der Prozessinstanz überprüft. Sie darf nur dann beendet werden, wenn sie sich im Zustand „decision making“ in der Workflow-Engine befindet. Wenn

der Zustand tatsächlich „decision making“ ist, erfolgt ein Zustandsübergang vom „decision making“ zu „end treatment“. Die Instanz wird dann beendet, das Arbeitsverzeichnis der Prozessinstanz aufgeräumt und eine HTTP-Antwort mit dem Status Code „200 OK“ an den Sender geschickt. Andernfalls empfängt der Sender einer Fehlermeldung mit Status Code „409 Conflict“.
Die zweite Methode bietet dem lokalen HCIS die Funktionalität, das Warten auf die externen initiierten Behandlungsprozesse einzustellen. Dabei wird die Prozessinstanz vom Wartezustand zum „decision making“ Zustand überführt. Wenn jedoch auf diese Prozessinstanz bezogenen Dokumente ankommen, erkennt die Prozess-Logik, dass die Prozessinstanz im „decision making“ Zustand überführt wurde. Der Sender empfängt dann eine Fehlermeldung mit Status Code „409 Conflict“.

7.4.2 Die HCIS API

Zur Übermittlung von Informationen vom DMPS an das HCIS werden noch zwei Schnittstellen generisch beschrieben, die vom HCIS als REST Webservices oder als einfache Servlets implementiert werden können. Die erste ist über den URI-Pfad */ExternalDocumentsNotification* ansprechbar und dient zur Notifikation bei der Initiierung neuer Behandlungsprozesse. Die API erwartet zwei URI Parameter: die Instanz-ID identifiziert die Prozessinstanz und die Mandatory-ID identifiziert den Beauftragte in der Gesundheitsorganisation. Dazu werden auch die empfangenen CDA-Dokumente im Body der Nachricht an das HCIS übermittelt. Die zweite Schnittstelle ist über den URI-Pfad */ProcessErrorsNotification* zu referenzieren und informiert über Fehler beim Prozessablauf. Die API erwartet die Instanz-ID und den Fehler als Parameter. Die folgende Tabelle verschafft noch mal eine Übersicht über die APIs.

System	HTTP-Methode	URI-Pfad	Beschreibung
DMPS	POST	*/reqrep*	empfängt Anfragen zur Initiierung internen Behandlungsprozesse
DMPS	POST	*/initiateExtTreatment*	initiiert externe Behandlungsprozesse
DMPS	POST	*/{instanceid}/initiateExtTreatment*	initiiert externe Behandlungsprozesse
DMPS	GET	*/{instanceid}/endTreatment*	beendet Prozessinstanzen
DMPS	GET	*/{instanceid}/escapeExtTreatment*	beendet das Warten auf die initiierten externen Behandlungen
HCIS	POST	*/ExternalDocumentsNotification*	empfängt Notifikation über neue initiierte Prozesse
HCIS	GET	*/ProcessErrorsNotification*	empfängt Meldungen über Fehlern im Prozessablauf

Tabelle 2: Komplete Übersicht über die vorgestellten APIs

7.5 Testkonzept und Testfälle

Weiterhin gehören zum Systementwurf die Festlegung der Testdaten und der Testfälle. Um das System möglichst vollständig und in reellen Bedingungen testen zu können, wird in dieser Arbeit ein Testszenarium entwickelt, das die wichtigsten Fälle in denen das System zum Einsatz kommen kann, abdeckt. Wie schon in früheren Kapiteln erwähnt wurde, kann ein verteilter Behandlungsprozess bezüglich der Reihenfolge der teilnehmenden Organisationen kettenförmig, sternförmig oder als eine Kombination von beiden ablaufen (vgl. **Abbildung 21**). Deswegen muss das Testszenarium diese möglichen Abläufe abdecken.

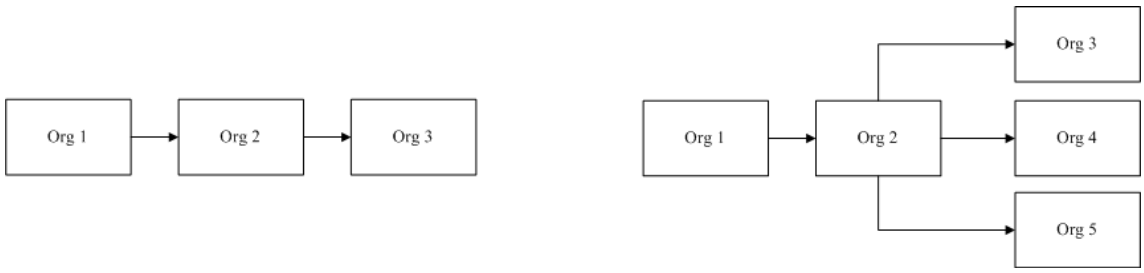


Abbildung 21: Mögliche Ablaufformen bei einem Behandlungsprozess

Hierbei muss berücksichtigt werden, dass in einem realen Fall die Interaktion zwischen DMPS-DMPS und non-DMPS-DMPS erfolgen kann und die Systeme als autonome Knoten existieren und miteinander kommunizieren, d.h. die Systeme im Testfall müssen vollständig autonom voneinander agieren. Im folgenden Szenarium (vgl. **Abbildung 22**) nehmen vier autonome Knoten teil, wobei die Struktur des Ablaufs das Minimum der möglichen Ablaufszenarien abdeckt.

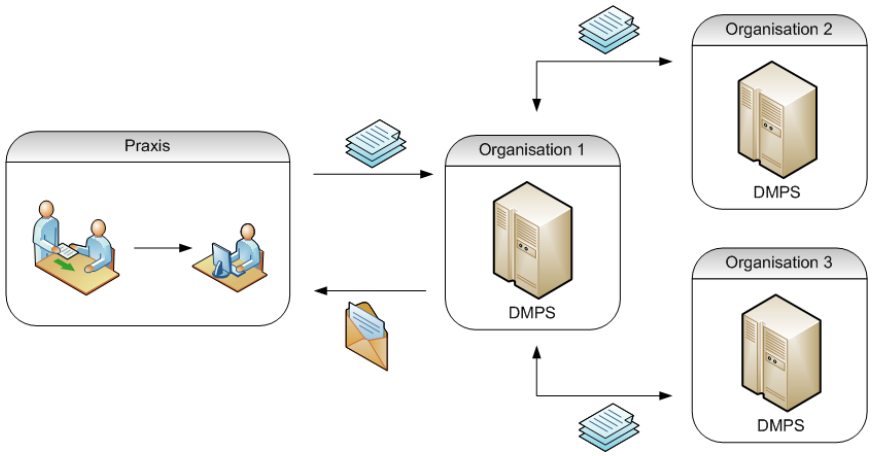


Abbildung 22: Ein minimales Szenarium zum Systemtest

Ein niedergelassener Arzt, der über das DMPS nicht verfügt, initiiert eine verteilte Behandlung. Der Arzt benutzt das angebotene Webportal von Organisation 1 und sendet ein CDA-Dokument an diese. Der Behandlungsprozess in Organisation 1 läuft intern, bis entschieden wird, dass der Patient zwei externe parallel verlaufende Behandlungen

in zwei autonomen Knoten benötigt. Das DMPS in Organisation 1 sendet dann parallel an Organisation 2 und Organisation 3, die über das DMPS verfügen, die CDA-Dokumente und die benötigten Prozesszustandsdokumente. Die parallelen Prozesse in Knoten 2 und 3 werden beendet und die Dokumente an Knoten 1 zurückgeschickt. Nach der Beendigung der Behandlung schickt Organisation 1 ebenfalls die resultierenden Dokumente an den Arzt.

Ein Vorteil von REST ist der mögliche offline Transport von Dokumenten ohne die Kenntnis von dem Webservices-Endpunkt des Empfängers. Der Patient kann die PKI-verschlüsselten Daten auf einem USB-Stick kriegen und den USB-Stick in die Radiologie mitbringen (vgl. **Abbildung 23**). Das HCIS in dieser Institution kann weiterhin die Daten aus dem Stick über die API ins DMPS einspeisen.

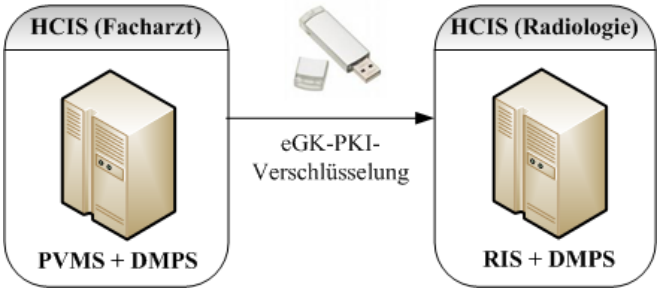


Abbildung 23: Offline Transport von PKI-verschlüsselten Dokumenten auf einem USB-Stick

8 Implementierung

Nach der Erstellung des System-Entwurfs wird in diesem Kapitel die eigentliche Implementierung des Anwendungssystems beschrieben. Das Ergebnis dieser Phase ist der in den vorangegangenen Kapiteln geplante Prototyp des Informationssystems für die Unterstützung von verteilten Behandlungsprozessen. Die Beschreibung erfolgt nach einzelnen Schwerpunkten getrennt und umfasst sowohl Vorgehensweise und Methoden als auch Ergebnisse. Die einzelnen Teile dieser Beschreibung umfassen Ausführungen zur jBPM Datenbankkonfiguration und Datenbankanbindung, über die Schnittstellen-Implementierung und das Webportal. Im Anschluss erfolgt eine detaillierte Beschreibung der Workflow-Implementierung für die Anwendung.

8.1 Die jBPM Datenbank

Die Hauptaufgabe der Persistenz ist es, die Prozesszustände während der Wartezustände sowie die Prozessdefinitionen zu speichern. Als Persistenzmechanismus verwendet jBPM Hibernate und ist damit von der eingesetzten Datenbank relativ unabhängig. Man darf jede Datenbank benutzen, die von Hibernate unterstützt wird. In dieser Arbeit wird die Derby-Datenbank von Apache verwendet. Im Folgenden werden die Schritte zur Anbindung, Konfiguration und Erzeugung der jBPM-Schemata mit dieser Datenbank erklärt.

8.1.1 Anbindung der Derby-DB und Konfiguration

Apache Derby ist ein Projekt der Apache Software Foundation. Derby basiert auf Java, wurde als ein relationales DBMS entwickelt und gehört wegen der einfachen Installation und der kleinen Installationsgröße zu den „schlanken“ Datenbanken. Auf die DB wird via Java Database Connectivity (JDBC) zugegriffen (vgl. **Abbildung 24**). JDBC bietet in der Regel eine einheitliche Schnittstelle zwischen der Datenbank und der Anwendung, die auf die DB zugreifen will. Die passenden JDBC-Bibliotheken *derby.jar* und *derbyclient.jar* müssen im Classpath vorhanden sein.

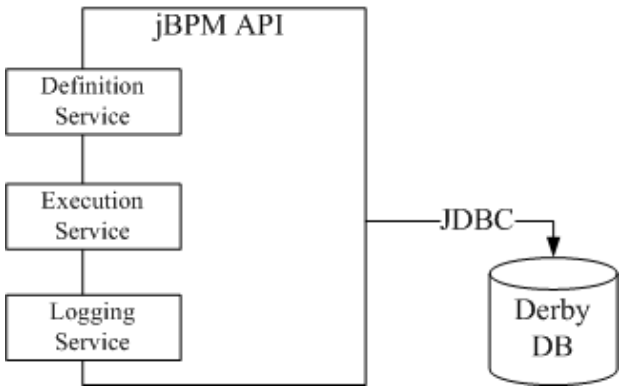


Abbildung 24: jBPM spricht Derby über JDBC an

Hibernate speichert die von jBPM benutzten Konfigurationen, in der *hibernate.cfg.xml* Konfigurationsdatei. Die Konfiguration muss bezüglich der verwendeten DB in diesem Fall DerbyDB angepasst werden. Die erste Zeile zeigt den verwendeten Hibernate-Dialekt. Es folgt die JDBC-Derby-Treiberklasse für die Verbindung zu der Datenbank. Die letzten drei Zeilen enthalten die Informationen zur Verbindung mit der leeren Derby-Datenbank für jBPM: die Verbindung-URL und die Login-Informationen.

```
<property name="hibernate.dialect">
      org.hibernate.dialect.DerbyDialect </property>
<property name="hibernate.connection.driver_class">
      org.apache.derby.jdbc.EmbeddedDriver </property>
<property name="hibernate.connection.url">
      jdbc:derby://localhost:1527/jbpm32 </property>
<property name="hibernate.connection.username">jbossjbpm</property>
<property name="hibernate.connection.password">jbossjbpm</property>
```

8.1.2 Die jBPM DB-Schemata

jBPM bietet die Möglichkeit, die DB-Schemata über die *jbpmConfiguration*-Klasse automatisch zu generieren. Die folgende Methode enthält den Kode zur Generierung der benötigten DB-Schemata:

```
public void createDBSchema() {
        JbpmConfiguration jc = null;
        DbPersistenceServiceFactory dbp = null;
        JbpmContext jcxt;

        jc = JbpmConfiguration.getInstance();
        dbp = (DbPersistenceServiceFactory)
       jc.getServiceFactory(Services.SERVICENAME_PERSISTENCE);
        dbp.createSchema();
}
```

8.2 Die DMPS Schnittstellen

Nachdem im Kapitel 7 die Schnittstellen des Systems als Funktionalität geschildert wurden, findet hier die eigentliche Implementierung dieser Funktionalitäten statt. Als Entwicklungsumgebung wurde die „NetBeans“ IDE verwendet, die hervorragende Funktionalitäten zur Entwicklung von RESTful Webservice mittels JSR 311 – die Java API für RESTful Webservices (JAX-RS) und Jersey bietet. Zusätzlich zur Erstellung von RESTful Webservices unterstützt die IDE das Testen, die Entwicklung von Client-Anwendungen, die auf die Webservices zugreifen sowie die Kode-Generierung zum

Aufruf von Webservices. Im Folgenden wird die Implementierung der einzelnen Schnittstellen der Workflow-Anwendung mit dem Einsatz von Jersey beschrieben.

8.2.1 Die „Accept External Documents“-API

Die API erwartet die HTTP-Anfragen über den relativen */reqrep* URI-Pfad.

```
@Path("/reqrep")
public class req_rep {
       @POST
       @ConsumeMime({"multipart/form-data",
       "application/xml", "multipart/mixed"})
       @ProduceMime("text/html")
       public Response MIMEPostRequest(@Context
       HttpServletRequest request)
}
```

Die Haupt-Methode der Klasse verarbeitet die POST Methode des HTTP-Protokolls und bietet die Kernfunktionalität zum Empfang von Dokumenten. Die @ConsumeMime -Notierung spezifiziert drei MIME Media Typen von Repräsentationen, die von der Methode als empfangene Ressourcen vom Client verbraucht werden können. Wie der Name besagt, wird der *"multipart/form-data"*-Typ zur Übergabe von Form-Daten benutzt. In der Anwendung wird der Typ zur Übergabe von Files über das entwickelte Webportal des DMPS verwendet. Der *"application/xml"*-Typ ist ein generischer Media-Typ für XML-Dokumente. Der letzte *"multipart/mixed"*-Typ unterstützt Multipart-Messages. Die einzelnen Teile der Nachricht werden durch eine Grenzlinie abgegrenzt. Die unterstützten MIME Media Typen decken alle Fälle ab in denen diese API geraten kann. Einerseits können Dokumente über das Webportal mit dem *"multipart/form-data"* übertragen werden. Andererseits erlaubt der *"multipart/mixed"*-Typ das Versenden von mehreren Dokumenten gleichzeitig in einer Nachricht. Mit dem *"application/xml"*-Typ können die einzelnen Dokumente auch separat versandt werden. Die @ProduceMime-Notierung spezifiziert dass eine *"text/html"*-Repräsentation als Antwort von der Ressource generiert und zum Client versendet wird.
Nachdem die API eine POST-Anfrage empfängt, werden mit der folgenden Funktion, die einzelnen Dokumente aus der Multipart-Nachricht gewonnen und in dem input-Folder gespeichert.

```
boolean processMultipartMIMERequest(HttpServletRequest request);
```

Zur Validierung der einzelnen CDA-Dokumente und der Prozesszustandsdokumente wird die XML_Processor-Klasse (vgl. **Abbildung 25**) verwendet. Die Klasse benutzt den DOM-Parser zum Parsing und zur Validierung. Die DMP_XML_Processor-Klasse erweitert die Funktionalitäten der XML_Processor-Klasse und bietet die Möglichkeit zum Processing von Prozesszustandsdokumenten. Neben der Möglichkeit, die einzelnen Elemente und deren Attribute zu lesen und zu manipulieren, bietet diese Klasse noch Methoden zum Merging von Historien in Prozesszustandsdokumenten.

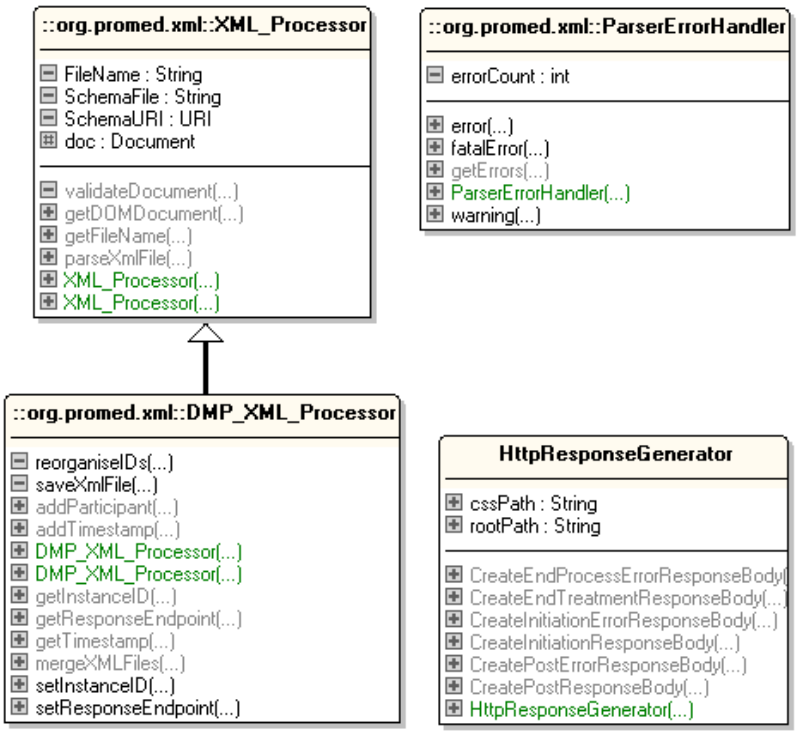


Abbildung 25: Das org.promed.xml Paket

Nach der Validierung generiert die Methode über die jBPM API eine Prozessinstanz in der Workflow-Engine und initialisiert die Instanz-Variablen.

```
private long generateProcessInstance(String InstanceID, String
ResponseEndpoint);
```

Die FS-Verzeichnisstruktur der Prozessinstanz wird dann erstellt und die Ausführung des Prozesses gestartet.

```
private void startProcessInstance(long instanceId);
```

Die Response-Nachrichten zur POST-Anfragen und GET-Anfragen werden von der HttpResponseGenerator-Klasse in **Abbildung 25** generiert.

8.2.2 Die „InitExtTreatment“-API

Die zwei Methoden zur Initiierung von externen Behandlungsprozessen sind über die relativen */initiateExtTreatment* und */{instanceid}/initiateExtTreatment* URI-Pfade zugänglich.
Die Methoden verarbeiten ebenfalls die POST Methode des HTTP-Protokolls und implementieren die Fälle zur Prozessinitiierung aus Kapitel 7. Als @ConsumeMime-Notierung sind drei MIME Media Typen wie bei der vorherigen API spezifiziert. Als @ProduceMime wird eine *"text/html"*- Repräsentation als Antwort generiert und zum Client versendet. Der erste Unterschied zwischen den beiden Methoden ist in der Path-

Notierung. Im ersten Fall wird über *"/initiateExtTreatment"* durch das HCIS ein neuer externer Behandlungsprozess gestartet indem eine neue Prozessinstanz initialisiert und gestartet wird, wobei der Prozess direkt im „extern treatment process“ Zustand überführt wird.

```
@Path("/initiateExtTreatment")
@POST
@ConsumeMime({"multipart/form-data", "application/xml",
"multipart/mixed"})
@ProduceMime("text/html")
public Response initiateExtTreatment(@Context HttpServletRequest
request);
```

Bei einer bereits laufenden Prozessinstanz wird im zweiten Fall eine weitere externe Behandlung initiiert. Dies geschieht über den */{instanceid}/initiateExtTreatment* URI-Pfad. Hier wird die Möglichkeit zur Einbettung der instanceid-Variable in der URI-Syntax benutzt. Die Variable ist mit der @PathParam-Notierung als Methode-Parameter in der initiateExtTreatment-Methode deklariert, und darf im Kontext dieser Methode verwendet werden, d.h. wenn der Benutzer die *.../123456/initiateExtTeatment* URI aufruft, extrahiert die Logik die Variable aus der Syntax und übergibt sie der Methode als Parameter.

```
@Path("/{instanceid}/initiateExtTreatment")
@POST
@ConsumeMime({"multipart/form-data", "application/xml",
"multipart/mixed"})
@ProduceMime("text/html")
public Response initiateExtTreatment(
       @PathParam("instanceid") String id,
       @Context HttpServletRequest request)
```

Die Response-Nachrichten zur POST-Anfragen bei den beiden Methoden werden ebenfalls mit Hilfe der HttpResponseGenerator-Klasse (vgl. **Abbildung 25**) generiert.

8.2.3 Die „End Treatment“-API

Das DMPS bietet zwei Wege zur Beendigung von Behandlungsprozessen. Zum einen kann man über den */{instanceid}/endTreatment* URI-Pfad eine im DMPS laufende Behandlung (Prozessinstanz) beenden. Die REST-Methode erwartet eine GET-Anfrage sowie die Instanz-ID der zu beendenden Instanz eingebettet in der URI-Syntax. Als eine Rückmeldung wird ebenfalls mit der @ProduceMime-Notierung eine *"text/html"*-Repräsentation spezifiziert und mit der HttpResponseGenerator-Klasse (vgl. **Abbildung 25**) von der Ressourcen generiert.

```
@Path("/{instanceid}/endTreatment")
@GET
@ProduceMime("text/html")
public Response endTreatmentProcess(
       @PathParam("instanceid") String id,
       @Context HttpServletRequest request)
```

Zum anderen kann man über den */{instanceid}/escapeExtTreatment* URI-Pfad eine im DMPS laufende Behandlung (Prozessinstanz), die sich im „extern treatment running"-Wartezustand befindet, d.h. die Instanz wartet noch auf initiierte externe Behandlungen, in den „decision making" Zustand überführen. Das bedeutet, dass das Warten auf die Behandlungen eingestellt wird.

```
@Path("/{instanceid}/escapeExtTreatment")
@GET
@ProduceMime("text/html")
public Response endTreatmentProcess(
       @PathParam("instanceid") String id,
       @Context HttpServletRequest request)
```

8.2.4 Die HCIS APIs

Zur Rückwärts-Kommunikation mit dem HCIS werden an dieser Stelle zwei generische APIs spezifiziert. Die erste „External Documents Notification"-API (vgl. Kapitel **7.4.2**) dient zur Notifikation bei Initiierung neuer Behandlungsprozesse. Bei der Notifikation sind zwei wichtige Information mit dem HCIS zu kommunizieren - die Instanz-ID und die Mandatory-ID, die den Beauftragten in der Gesundheitsorganisation identifiziert. Diese Informationen können als URI Parameter übertragen werden. Zusätzlich sollen auch die vom DMPS empfangenen CDA-Dokumente im Body der Nachricht an das HCIS übermittelt werden. Deswegen soll die API die Möglichkeit zur Verarbeitung von POST-Anfragen implementieren. Die API ist über den */ExternalDocumentsNotification* URI-Pfad zugänglich.
Die zweite Schnittstelle ist über den */ProcessErrorsNotification* Pfad zu referenzieren und informiert über Fehler beim Prozessablauf. Sie erwartet die Instanz-ID und den Fehler als Parameter. Da die API keine Dokumente übermitteln soll, kann diese als GET-Methode implementiert werden.
Die APIs können als REST-Methoden mit Jersey oder als einfache Servlets implementiert werden können. Eine mögliche Spezifikation ist im Anhang **A.2** zu finden.

8.2.5 Das DMPS-Webportal

Die REST-API des Systems realisiert die funktionale Anforderung zur Interaktion zwischen Institutionen mit integrierten DMP-Systemen. Neben dieser Anforderung soll das System die Möglichkeit bieten, dass auch non-DMPS-Anwender über ein Webportal mit dem Zielsystem kommunizieren können. Der Benutzer ist in der Lage über einen Browser, dem DMPS, Dokumente zu übertragen und externe Behandlungen

einzuleiten. Das Webportal wird auf Basis der JSP-Technik implementiert. Der Benutzer kann mittels Webformulare die benötigten Informationen ausfüllen und diese absenden. Die Information wird dann von der Logik bearbeitet und als GET- oder POST-Anfrage direkt an die DMPS-APIs versendet.

8.3 Der DMPS-Workflow

Nachdem in der Entwicklung schon eine konkrete Workflow-Engine betrachtet wird, wird hier mit den Mitteln seiner Workflow-Sprache das Workflow-Model formuliert und der Workflow implementiert. Man beachte, dass die Entscheidung für eine Workflow-Engine auch zum Gebrauch seiner speziellen Workflow-Sprache zwingt. Das gewonnene diagnostisch-therapeutische Modell aus Kapitel **7.1**, bildet jetzt den Ausgangspunkt für die Spezifikation des Workflow-Modells.
Wie schon angedeutet führt jBPM die Prozesse aus, die in jPDL in XML formuliert sind. Zur Modellierung kann man auch der grafische Prozess-Designer in Eclipse verwenden. Die folgenden Prozesskonstrukte kommen bei der Modellierung zum Einsatz: die Start- und End-States repräsentieren die Start- und Endknoten des Modells. Bei den Entscheidungsknoten werden die Bedingungen für die Austritt-Transitionen überprüft. Die erste Transition für welche die Bedingung erfüllt ist, wird genommen. Die State-Nodes sind einfache Wartezustand-Knoten. Sie sind zu benutzen, wenn der Prozess auf ein externes System warten muss. Beim Knoteneintritt könnte z.B. eine Nachricht an das externe System gesendet werden nachdem der Prozess im Wartezustand bleibt. Wenn das externe System eine Rückmeldung schickt, könnte dies zur token.signal()-Funktion führen, was die weitere Prozessausführung triggert. Diese Art von Knoten werden bei der Modellierung am meisten benutzt, denn der Prozess sehr oft mit externen Systemen interagieren muss. Das letzte Element ist der einfache Knoten, in welchem man einen eigenen Code schreiben kann. Dies ist der Fall, wenn z.B. die funktionale Logik mit der JavaAPI implementiert werden muss. Der Designer bietet noch andere wichtige Konstrukte, die oben genannten reichen aber zur Beschreibung des folgenden Workflow-Modells aus. Zur Realisierung der fachlichen Logik nutzt die Anwendung jBPM auf zwei Arten:

- Aufruf von jBPM durch die Anwendung: Im Ablauf müssen die einzelnen Komponenten Entscheidungen treffen und manuell eingreifen. Ein Fall wäre wenn die API-Logik die Ausführung einer Prozessinstanz im Wartezustand fortsetzt oder eine neue Prozessinstanz initialisiert.
- Aufruf der Fachlogik durch jBPM: Die Logik wird durch automatisch getriggerten Aktionen, wie z. B. durch Events oder Entscheidungen aufgerufen. In einem solchen Fall ruft die jBPM-Engine, die in der Prozessbeschreibung genannte Handler-Klasse auf. Dieser Mechanismus wird auch „delegation“ genannt. Ein Beispiel folgt weiter unten.

Die jPDL bietet dazu die Möglichkeit ActionHandler-Klassen direkt an Decision-Nodes, Task-Nodes oder andere Elemente zu hängen. Im folgenden Beispiel ruft die Engine den Handler auf, sobald der Node „document with valid ID“ erreicht wird:

```
<node name="document with valid ID">
	<event type="node-enter">
		<action class="org.promed.actions.ResumeExistingInstance"
		name="ResumeExistingInstance"></action>
	</event>
	<transition to="treatment ready" name="to treatment
	ready"></transition>
</node>
```

Eigene Action-Klassen müssen das „ActionHandler“-Interface mit der Methode „execute()“ implementieren:

```
public class ResumeExistingInstance implements ActionHandler {
	private static final long serialVersionUID = 1L;
	public void execute(ExecutionContext context) throws Exception {
		....
	}
}
```

Die Verbindung zwischen Ablauf und Action-Klassen sind direkt „hart verdrahtet“ in der DMP-Prozessdefinition dokumentiert. Diese Klassen, sowie alle Handler-Klassen des Workflows als UML-Diagramme sind im Anhang **A.4** dieser Arbeit zu finden.
Das DMP-Modell in **Abbildung 26** wurde mit dem jPDL Process Designer in Eclipse modelliert und schildert die Kernfunktionalität der entwickelten Plattform. Die Prozessdefinition in XML ist im Anhang **A.3** zu finden. Wie aus der Darstellung ersichtlich, beginnt der Prozess mit einem Start-Knoten wenn patient-bezogene Information in Form von CDA-Dokumente von der API empfangen werden. Die „Accept External Documents“-API hat also eine neue Prozessinstanz gestartet und die empfangenen Dokumente im zugehörigen Verzeichnis gespeichert. Es folgt eine Entscheidung im Decision-Knoten, die die Ablaufstruktur in zwei Graphen verzweigt. Der Entscheidungsprozess im Knoten bezieht sich auf die empfangene Information im Prozesszustandsdokument. Die Handler-Klasse muss das „DecisionHandler“-Interface mit der folgenden Methode implementieren:

```
public class DecisionMakerExistsId implements DecisionHandler {
	private static final long serialVersionUID = 1L;

	public String decide(ExecutionContext executionContext);
}
```

Der Rückgabe-Wert der Methode enthält die Austritt-Transition die genommen werden muss.
Die Logik hinter dem Knoten vergleicht die IDs der gestarteten Prozessinstanzen in der Datenbank mit der Instanz-ID im Dokument. Der Vergleich erfolgt mit der „findInstance()“-Methode aus der „ControlProcess“-Klasse (vgl. **Abbildung 36** im Anhang). Man unterscheidet nach der Entscheidung zwischen zwei Fälle:

- Das System findet keine Übereinstimmung mit der Instanz-ID, die in dem Prozesszustandsdokument enthalten ist. Dies ist der Fall wenn ein externer Partner eine externe Behandlung einleiten will.
- Das System findet eine Übereinstimmung mit der Instanz-ID, die in dem Prozesszustandsdokument enthalten ist. Dies ist der Fall wenn die Resultate aus einer erfolgten externen Behandlung in Form von CDA-Dokumente an das DMPS zurück geschickt werden.

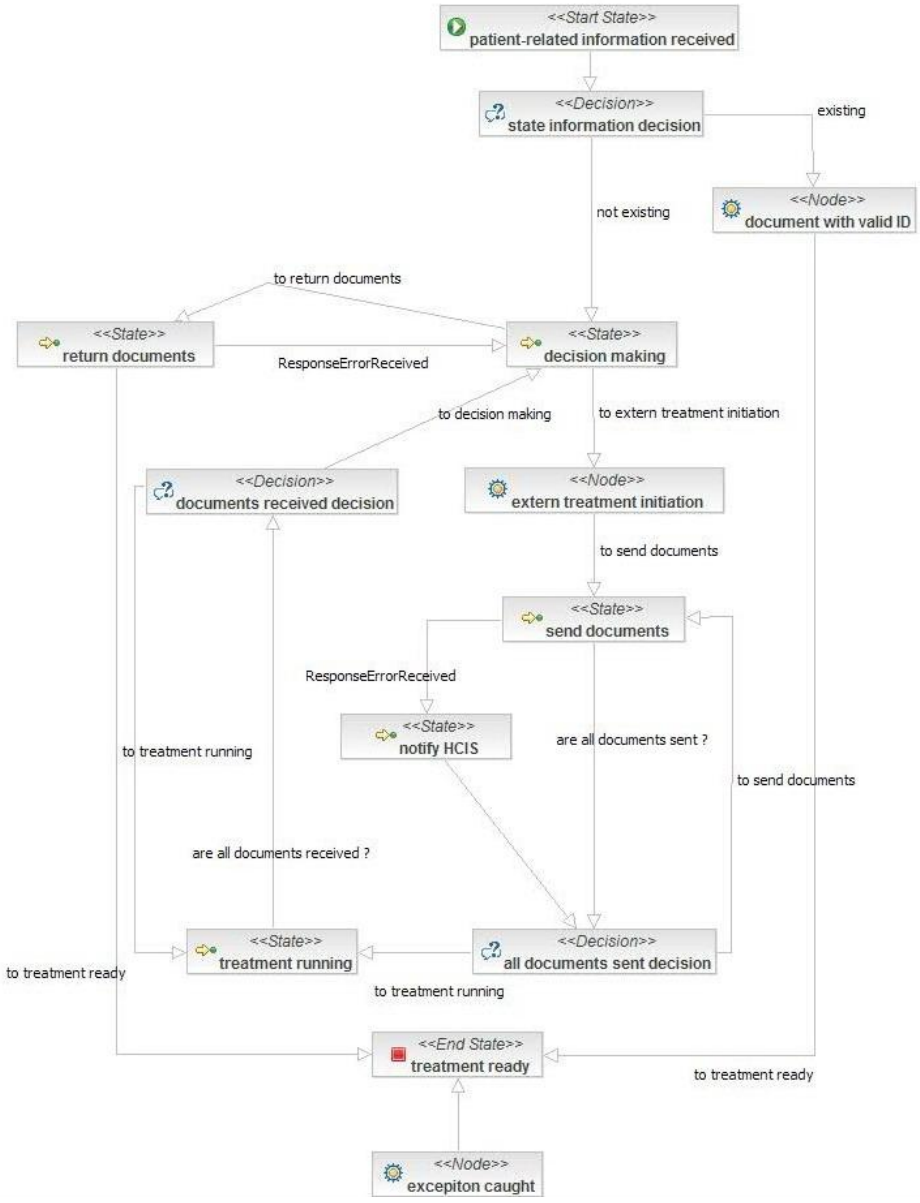


Abbildung 26: Das Workflow-Model des DMPS

Im ersten Fall wird das HCIS über den Empfang der Dokumente, d.h. über die neue Behandlungsinitiierung benachrichtigt. Die Empfang-Notifikation erfolgt während des Zustandsübergangs zum „decision making"-Knoten. An der Transition zum „decision making" ist eine Aktion angehängt, die beim Zustandsübergang ausgeführt wird. Die

Aktion sendet dann mittels der „POSTHttpRequest“-Hilfsklasse (vgl. **Abbildung 36** im Anhang) eine POST-Nachricht an das HCIS. Mit dieser Nachricht werden auch die empfangenen Dokumente aus dem Instanz-Verzeichnis übertragen.
Der Prozess bleibt solange im Wartezustand während dieser Phase bis eine Nachricht vom HCIS über die API zur Fortsetzung der Instanz empfangen wird. Diese Nachricht kann zwei verschiedene Zustandsübergänge triggern. Zum einen können die lokalen Systeme über die angebotenen Schnittstellen den Behandlungsprozess beenden. Die Prozessausführung wird dann im „return documents“ Zustand gesetzt (vgl. **Abbildung 27**).

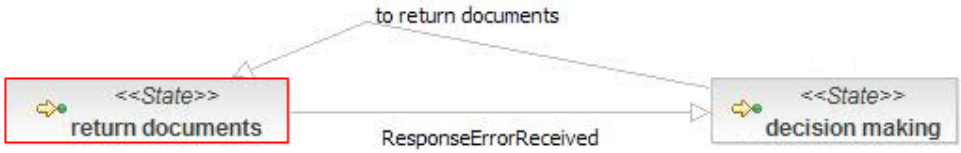


Abbildung 27: Vor der Beendigung einer Prozessinstanz müssen die Resultate aus der Behandlung zurückgeschickt werden

Die vom HCIS übergebenen CDA-Dokumente werden an die ResponseEndpoint-URI aus dem Prozesszustandsdokument zurückgeschickt. Der Prozess bleibt wartend bis eine Acknowledgement-Antwort empfangt wird. Im Falle eines Fehlers bei der Übertragung, wird das HCIS über die „Error Notification“-API benachrichtigt und die Prozessinstanz in den „decision making“-Zustand überführt. Im Erfolgsfall wird die Prozessinstanz beendet.
Zum anderen können beliebig viele parallele externe Behandlungen vom HCIS durch eine POST-Anfrage über die „InitExtTreatment“-API initiiert werden. Die Logik generiert zuerst im „extern treatment initiation“-Zustand ein Prozesszustandsdokument (vgl. **Abbildung 28** links).

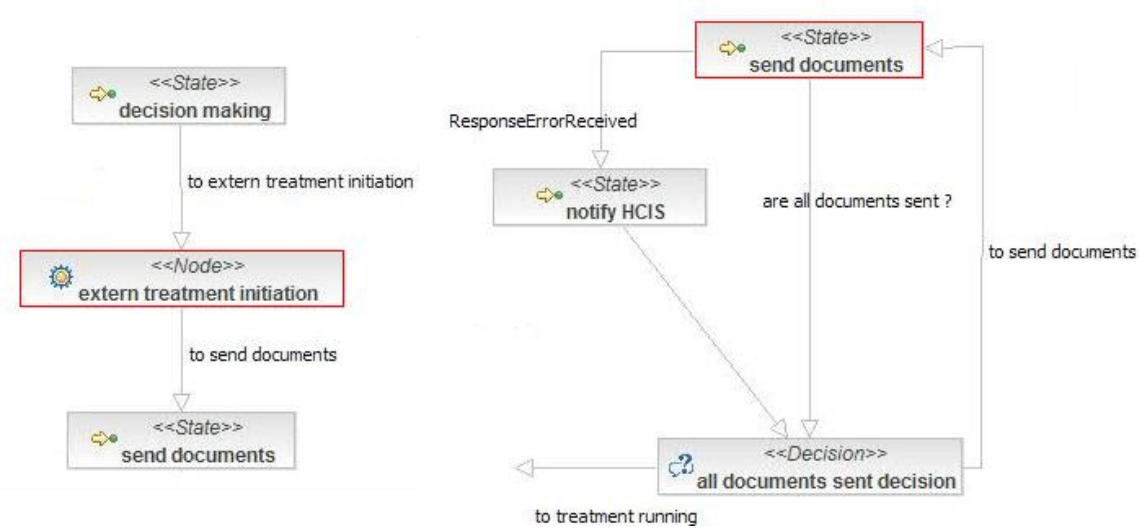


Abbildung 28: Teil des DMPS-Workflows zur Initiierung einer externen Behandlung

Das Dokument enthält die ID der momentanen Prozessinstanz, die URI der „Accept External Documents“-API als Response-Endpoint und die momentane Institution als

Eintrag in der Historie. Nach der Generierung des Dokuments, wird der Prozess in den „send documents“ Zustand überführt. Aus diesem Zustand beginnt eine Schleife, die die Aufgabe hat, die CDA-Dokumente und das Prozesszustandsdokument sequenziell an allen vom HCIS identifizierten Empfänger zu versenden (vgl. **Abbildung 28** rechts). Dieser Ablauf gliedert sich in vier wichtige Schritte. Als erstes schickt die Fachlogik hinter dem „send documents“-Knoten die Dokumente an den ersten identifizierten Empfänger. Die Prozessinstanz wartet in diesem Zustand auf eine Bestätigung vom Empfänger. Dieser Knoten hat zwei ausgehende Transitionen, die bezüglich der Rückmeldung vom Empfänger möglich sind. Im Falle einer Fehler-Rückmeldung versucht die Logik noch zweimal die Dokumente zu versenden. Bei einem Misserfolg wird das HCIS über die erfolgslose Kommunikation mit dem Endpoint benachrichtigt und die Schleife weiter ausgeführt.
Im Falle einer Bestätigung erfolgt eine Transition zum Entscheidungsknoten. In diesem Zustand überprüft die Logik, ob bereits alle Dokumente erfolgreich versand wurden. Nach der Entscheidung wird die Schleife entweder nochmals ausgeführt oder die Prozess-Ausführung in den „treatment running“-Wartezustand überführt, solange CDA-Dokumente und Prozesszustandsdokumente von allen externen Prozess-Teilnehmern zurückgeschickt werden.
Aus diesem Zustand folgt die logische Verbindung zu der zweiten großen Workflow-Verzweigung und zwar wenn die Logik eine Übereinstimmung der in dem Prozesszustandsdokument enthaltenen Instanz-ID mit einer ID in der Persistenz-Datenbank feststellt. Dies ist der Fall wenn die Resultate aus einer erfolgten externen Behandlung an eine sich im „treatment running“-Zustand befindende Prozessinstanz zurückgeschickt werden (vgl. Abbildung 31).

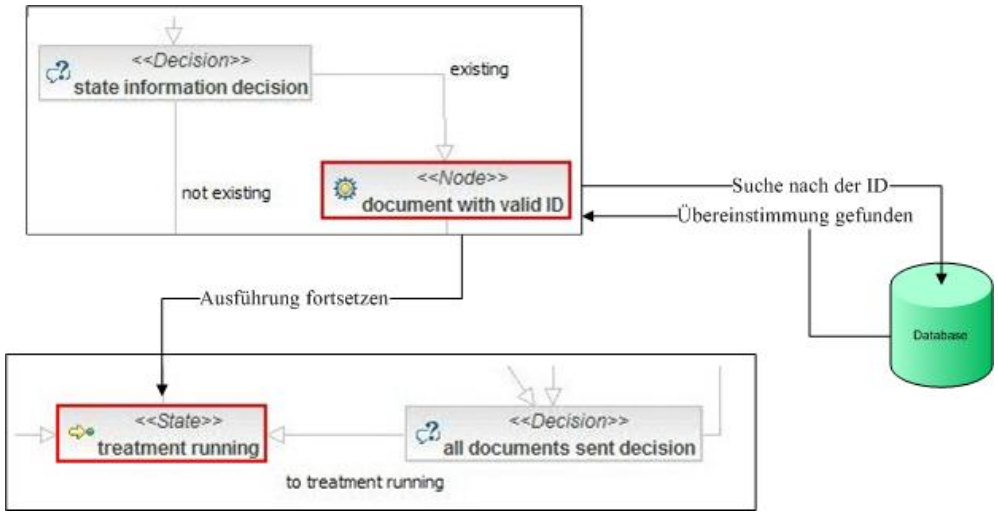


Abbildung 29: Die System-Logik findet über die empfangene ID die richtige Instanz in der Datenbank und setzt die Ausführung dieser Instanz vor

Das DMPS findet über die ID die richtige Instanz, die sich in einem „treatment running“-Wartezustand befindet, übergibt die empfangenen Dokumente dieser Instanz und setzt die Ausführung des Prozesses fort. Die andere Prozessinstanz wird dann

beendet. Wenn alle externen Behandlungen erfolgreich beendet und die Resultate empfangen sind, erfolgt eine Transition wieder zum „decision making“-Wartezustand. Der Prozess durchläuft diese Schleife beliebig oft, bis entschieden wird, dass der Behandlungsprozess beendet werden soll.
Da die Zuverlässigkeit des Systems aus Sicht der semantischen Fehler, die zu einem inkonsistenten Zustand führen können, eine wichtige Rolle spielt, wird im Workflow-Modell ein zusätzlicher Ausnahmebehandlungsmechanismus implementiert. Der Ausnahmebehandlungsmechanismus von jBPM betrifft nur Java-Ausnahmen, d.h. die Prozess-Ausführung kann keine Ausnahmesituationen aus Sicht von jBPM verursachen. Nur die Ausführung der Delegation-Klassen kann zu Ausnahmen führen. Für die Prozessdefinition, die Knoten und die Transitionen kann man Ausnahme-Handler spezifizieren, die bestimmte Aktionen ausführen, wenn Ausnahmen in den Delegation-Klassen eintreten. Es ist darauf hinzuweisen, dass eine vom jBPM abgefangene Ausnahme keinen Einfluss auf den Ablauf hat. Um den Kontrollablauf zu beeinflussen, besteht die Möglichkeit in der Aktion zum Ausnahme-Handler, über die jBPM-API die Ausführungsreihenfolge von einem Zustand in anderen Zustand zu ändern. Aus dieser Sicht wurden neben den Konstrukten und Situationen zur Unterstützung des zuverlässigen Ablaufs noch eine Ausnahme-Handler-Klasse und ein Ausnahme-Knoten (vgl. **Abbildung 30**) als Ausnahmebehandlungsmechanismus eingeführt. Diese Klasse ist zentral für den Workflow spezifiziert und fängt alle Ausnahmen verursacht durch die Delegation-Klassen ab. Der folgende Kode dieser Klasse ändert den Kontrollablauf ab und setzt die Ausführungsreihenfolge im Ausnahme-Knoten:

```
public class ProcessException implements ActionHandler{
       private static final long serialVersionUID = 1L;
       public void execute(ExecutionContext executionContext){
              Node exceptionNode =
              executionContext.getProcessDefinition().getNode("exception
              caught");
              executionContext.getToken().setNode(exceptionNode);
       }
}
```

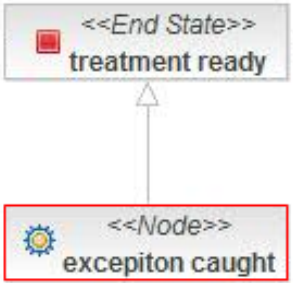


Abbildung 30: Der Ausnahme-Knoten benachrichtigt das HCIS über die Fehler und leitet die Ausführung zu dem Endzustand

Der Ausnahme-Knoten ist durch eine Transition direkt mit dem Endzustand-Knoten verbunden (vgl. **Abbildung 30**). Bei einer Ausnahme benachrichtigt dieser das HCIS

über die auftretenden Fehler und leitet die Ausführung zu dem Endzustand. Nach Beschreibung der Implementierung wichtiger Anwendungskomponente folgen im nächsten Kapitel die Systemkonfiguration und der Systemtest.

9 Systemkonfiguration

In diesem Teil der Studie werden die einzelnen Komponenten zu einem lauffähigen System verbunden und konfiguriert. Im Rahmen dieser Aufgabe, erfolgt zudem auch der Systemtest, bei dem das System bezüglich der im Kapitel **7.5** eingeführten Testfälle unter Betriebsbedingungen getestet wird.

9.1 Ablaufumgebung des Systems

jBPM ist auf der Process-Virtual-Machine (PVM) aufgebaut. Die PVM ist eine einfache Java-Bibliothek, die in allen Java-Umgebungen ausgeführt werden kann. Somit kann jBPM sowohl in einem stand-alone Programm oder Servlet-Container betrieben werden. Diese Tatsache und die Nutzung von REST Webservices als Kommunikationsmittel erlauben den Einsatz von einem Servlet-Container als Ablaufumgebung des Systems, der Apache Tomcat-Container. Tomcat ist eine Java-Referenzimplementierung der Servlet- und JSP-Technologie und bietet eine Ablaufumgebung für Java-basierte Webanwendungen. Zum besseren Verständnis der Systemkonfiguration mit Tomcat, wird mit der folgenden Tabelle 2 die Tomcat-Verzeichnisstruktur vorgestellt.

Folder	Beschreibung
/bin	Startskripte und BootstrapKlassen
/common/lib	Klassen für Tomcat und Webanwendungen
/conf	Konfigurationsfiles
/logs	Log-Information
/server/libs	Tomcat-interne Klassen
/webapps	Verzeichnis der Webapplikationen
/work	Arbeitsverzeichnis für die kompilierten Klassen, etc.

Tabelle 3: Die Tabelle stellt die Verzeichnissstruktur von Tomcat dar

9.1.1 jBPM auf Tomcat

Vor dem Einsatz von jBPM auf Tomcat als Ablaufumgebung für die Workflow-Engine, muss jBPM vorerst Tomcat-spezifisch angepasst werden. Als erster Schritt wird die jBPM Webanwendung nach der Java-Servlet-Spezifikation vorbereitet indem ein Web-Archiv generiert wird. jBPM bietet im „jbpm-jpdl-3.2.3/deploy"-Installationsfolder ein Ant-Task zur automatischen Generierung eines Web-Archives für Tomcat:

```
ant customize.console.for.tomcat
```

Es wird ein Archiv im „jbpm-jpdl-3.2.3/deploy/customized"-Verzeichnis generiert. Nun erfolgen die Derby-Einstellungen in der Hibernate-Konfigurationsdatei wie im Kapitel **8.1.1** beschrieben: (/WEB-INF/classes/hibernate.cfg.xml). Dazu müssen die Derby-JDBC-Bibliotheken nach dem „/common/lib"-Folder im Tomcat kopiert werden.

Abbildung 31 zeigt die von jBPM benutzten Bibliotheken, die im „/WEB-INF/lib“-Folder liegen müssen. Die Bibliotheken mit der fetten Schrift müssen zusätzlich zu den bestehenden Bibliotheken kopiert werden:

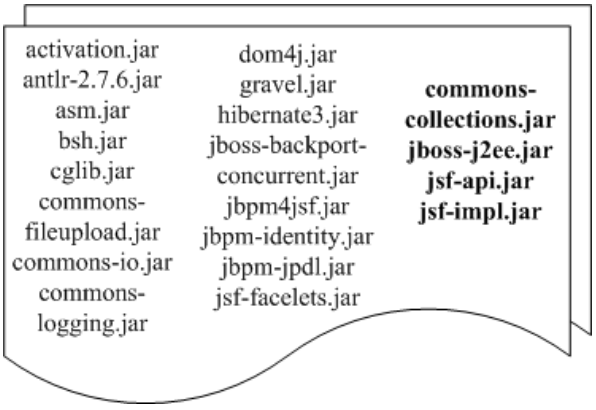


Abbildung 31: Die Kern-Bibliotheken von jBPM

- jboss-j2ee.jar (das Archiv ist in einer gewöhnlichen JBossAS-Distribution zu finden)
- commons-collections.jar (das Archiv ist in einer gewöhnlichen JBossAS-Distribution zu finden)
- jsf-api.jar und jsf-impl.jar (jsf-Bibliotheken von Sun)

Nach diesen Einstellungen ist nun das Webarchiv bereit zum Einsetzen in Tomcat. Dies geschieht indem das Archiv nach „/webapps“-Folder von Tomcat kopiert wird. Man kann jetzt die jBPM-Webapplikation starten, kann sich aber nicht erfolgreich einloggen. Der Grund dafür ist, dass das Security-Realm von Tomcat bezüglich der jBPM spezifischen Benutzer und Rollen nicht richtig konfiguriert ist. Durch das standard filebasierte Memory-Realm von Tomcat kann man die spezifischen Benutzer und Rollen im Bezug auf jBPM in der „/conf/tomcat-users.xml“-Datei einfügen:

```
<tomcat-users>
  <role rolename="manager"/>
  <role rolename="standard"/>
  <role rolename="admin"/>
  <role rolename="user"/>
  <user username="ernie" password="ernie" roles="user"/>
  <user username="admin" password="admin" roles= "manager,admin"/>
  <user username="user" password="user" roles="user"/>
</tomcat-users>
```

Nach dieser Konfiguration ist man in der Lage die Webapplikation zu starten und sich einzuloggen. Die Webapplikation bietet zahlreiche Funktionalitäten wie das Einsetzen von Prozessdefinitionen, die Steuerung der einzelnen Prozessinstanzen und die Rollen- und Benutzerverwaltung.

9.1.2 RESTful Webservices mit Jersey auf Tomcat

Die DMPS-Webservices werden als Servlets in der Tomcat-Umgebung eingesetzt und zwar separat und unabhängig von der Workflow-Engine. Die Kommunikation zwischen diesen Komponenten erfolgt explizit über die jBPM API und erfordert, dass die jBPM Kern-Bibliotheken und die Konfigurationsfiles bezüglich der Persistenzhaltung dem RESTful-Servlet zur Verfügung stehen. Zusätzlich müssen noch XSD-Schemata Files für die XML-Validierung sowie die JSP-Files für die Generierung der HTML-Masken verfügbar sein. Deshalb folgt vorerst eine Schilderung der Web-Archive-Struktur dieses Servlets nach der Java-Servlet-Spezifikation.

```
/index.jsp
/initiateExtTreatment.jsp
/images/...
/css/...
/schemata/...
/WEB-INF/web.xml
/WEB-INF/classes/hibernate.cfg.xml
/WEB-INF/classes/org/promed/ws/...
/WEB-INF/classes/org/ promed/xml/...
/WEB-INF/lib/...
/META-INF/MANIFEST.MF
```

Neben der vorgeschriebenen Datei „META-INF/MANIFEST.MF“ enthält jede WAR-Datei ein Verzeichnis „WEB-INF“. In diesem Verzeichnis befindet sich der *Deployment Descriptor* namens „web.xml“. Dieser definiert alle Servlets und andere Eigenschaften der Webapplikation. Die benötigten Bibliotheken, d.h. Jersey-, jBPM-, Hibernate-Bibliotheken etc., sind in „WEB-INF/lib“ enthalten. Die kompilierten Java-Klassen, die die eigentliche Logik verkörpern, sowie die Hibernate-Konfigurationsdatei werden in dem „/WEB-INF/classes“-Verzeichnis gespeichert. Außer dem WEB-INF-Verzeichnis ist der statische Inhalt der Webanwendung enthalten. Das sind die Graphiken, die Layout-Files, die XSD-Schemata und die JavaServer-Pages.
Wie schon erwähnt, wird in dem *Deployment Descriptor* alles bezüglich der Webapplikation außer dem Kontext-Pfad definiert, letzter wird beim Deployen festgesetzt. Für das DMPS wurde das Web-Archiv mit dem „/RESTServices“-Kontext-Pfad eingesetzt, jedoch kann der Administrator diesen Pfad beliebig festlegen. Da die Logik hinter den Webservices die eigentliche Initialisierung der Prozessinstanzen vollbringt, werden einige wichtige System-Konfigurationsparameter als Servlet-Kontext-Parameter in dem *Deployment Descriptor* definiert. Die Parameter sind dann für die Webapplikation sichtbar, jederzeit zugreifbar und müssen nur an einer Stelle in der „web.xml“-Datei gepflegt werden. Diese zentralisierten Systemeinstellungen begünstigen eine verständliche, nachvollziehbare Systemkonfiguration. Die folgende Tabelle schildert die einzigen Parameter und deren Bedeutung für das System:

Folder	Beschreibung
ProcessName	Der Name des Prozesses eingesetzt als Prozessdefinition in der Prozessdatenbank
DMPBaseURL	Die Basis-URL der DMPS REST APIs. Sie besteht aus der

	Container-Basis-URL und dem Servlet-Kontext-Pfad
DMPReqRepURL	Die DMPS REST API zum Empfang von Dokumenten, die als ResponseEndpoint in die Prozesszustandsdokumente gesetzt wird
OrgInfoURL	Die Entscheidung zum Eintrag in die Historie wird von jeder Institution separat getroffen. Aus diesem Grund stellt die OrgInfoURL eine Referenz zu der Info-Seite der teilnehmenden Healthcare-Organisationen zur Verfügung, die prozessbegleitend als Behandlungshistorie übermittelt werden kann.
DMPRepository	Der Repository-Pfad zur Persistierung der Dokumente
HCISBaseURL	Die Basis-URL der HCIS HTTP APIs.

Tabelle 4: Die zentrale Parameter für das DMPS werden als Servlet-Kontext-Parameter in dem *Deployment Descriptor* definiert

In **Abbildung 32** ist die physische Architektur des Systems aus Sicht der eingesetzten Ablaufumgebung dargestellt.

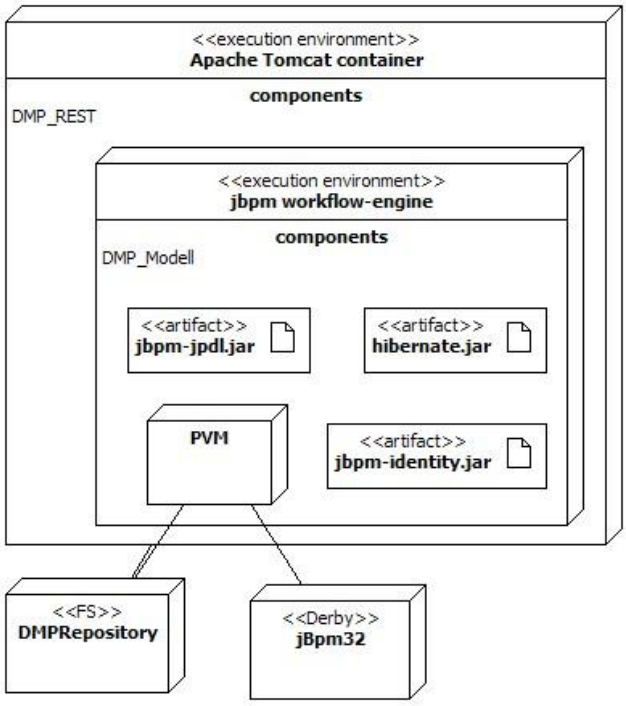


Abbildung 32: Die physische Architektur des Systems aus Sicht der eingesetzten Ablaufumgebung.

9.2 Systemtest

Teil dieser Arbeit ist neben der Systementwicklung auch das Testen der entworfenen Anwendung. Deswegen soll hier das Testszenarium aus Kapitel 7.6 unter möglichst realen Bedingungen simuliert und durchgeführt werden. Diese Anforderung impliziert dass die drei Systeme aus dem Testfall in Bezug auf die eingesetzte Ablaufumgebung, vollständig autonom voneinander laufen. Es ist eine häufig anzutreffende Anforderung

in Umgebungen, in der sich mehrere Entwickler oder Tester einen Entwicklungsserver teilen müssen, die sich aber nicht gegenseitig stören dürfen. Tomcat bietet hierzu die Möglichkeit auf einfache Weise die Software ein einziges Mal installieren und dann verschiedene, vollkommen getrennt konfigurierte Instanzen des Servers nebeneinander laufen lassen. Natürlich dürfen diese sich nicht bezüglich der Ressourcen wie Ports, Temp-Verzeichnisse usw. beeinflussen. Als Vorbereitung für die weiteren Tomcat-Instanzen erfolgt als erstes eine Isolation der ursprünglichen Tomcat-Instanz. Dafür werden die instanzspezifischen Files (in Conf, Temp, Logs, usw.) in einem Verzeichnis für die einzelnen Instanzen abgelegt ist. Die *catalina_home*-Variable stellt dabei das Root-Verzeichnis der Tomcat-Installation dar und währenddessen *catalina_base* das Root-Verzeichnis der jeweiligen Tomcat-Instanzen ist. Da dem Testfall zufolge drei autonome DMP-Systeme interagieren, werden hier logischerweise drei Tomcat-Instanzen mit den folgenden Schritten konfiguriert:

- Im Verzeichnis *catalina_home*, in diesem Fall „/apache-tomcat-6.0.18“, legt man drei Unterverzeichnisse für die Instanzen an: „/ServerA“, „/ServerB“ und „/ServerC“.
- In diesen drei Verzeichnissen werden nun Kopien der „/bin“, „/conf“, „/webapps“ aus der Originalinstallation abgelegt.
- Im „/bin“ werden verschiedene Skripte für die Tomcat-Instanz abgelegt. Die Startskripte startup.bat müssen bezüglich *catalina_base* und der neuen Instanz-Verzeichnisse angepasst werden:

```
set CATALINA_BASE = %CATALINA_HOME%\Server{A,B,C}
```

Nachdem die Verzeichnis-Strukturen für die Instanzen abgelegt sind, muss die Datei „server.xml“ für jede Instanz angepasst werden. Diese Datei, welche die Konfiguration des Tomcat-Servers als Ganzes beschreibt, ist bei der Installation nicht für eine solche geteilte Server-Umgebung angepasst. Dabei muss jeder Instanz eine einmalig vorkommende Portnummer für die Connectoren und Server zugewiesen werden. Zur Verdeutlichung folgt ein Auszug aus der Datei:

```
<Server port="~~8005~~ 8001" shutdown="SHUTDOWN" debug="0">
...
<Connector
port="~~8080~~ 8081" protocol="HTTP/1.1" connectionTimeout="60000"
redirectPort="8443"/>
...
 <Connector port="~~8012~~ 8011" protocol="AJP/1.3" redirectPort="8443" />
...
```

Zusätzlich zu den separaten Tomcat-Instanzen werden noch separate Dokument-Repositorien für die DMP-Systeme angelegt. In der folgenden Abbildung sind die wichtigsten DMPS-Konfigurationen für den Testfall dargestellt.

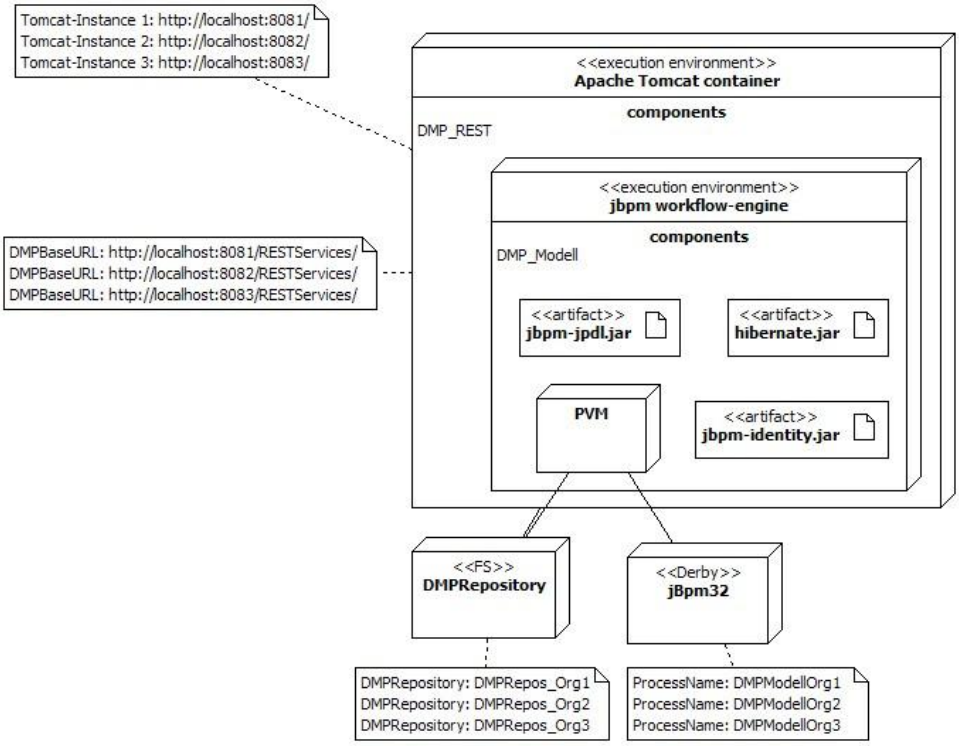


Abbildung 33: DMPS-Konfigurationen für die einzelnen Systeme

Nach diesen Schritten können die Tomcat-Instanzen gestartet, die DMPS-Komponente eingesetzt und die Webanwendungen getestet werden.

10 Evaluation

In diesem Kapitel wird auf die Evaluation der vorgestellten Software-Lösung in Hinsicht auf die gestellten Anforderungen eingegangen. Hierbei werden zunächst die funktionalen Anforderungen betrachtet und anschließend die nicht-funktionalen Anforderungen bewertet.

Im Rahmen dieser Arbeit wurde eine Plattform entwickelt und vorgestellt, die eine Unterstützung von verteilten Behandlungsprozessen bietet. Ein Prozess-Modell wurde als eine Erweiterung des klassischen diagnostisch-therapeutischen Zyklus modelliert und ein Prozesszustandsdokument zur Übertragung von Prozessinformationen und Behandlungshistorien instrumentiert. Mit dem RESTful Architekturstil wird sowohl die Internet-Kommunikation mit anderen DMP-Systemen als auch den Offline-Transfer von Information ermöglicht. Zudem können auch non-DMPS Benutzer die RESTful Webservices des DMPS über ein Webportal nutzen. Aus Benutzersicht decken diese Funktionalitäten alle gestellten Anforderungen an das DMPS ab.

Mit den lokalen HCI-Systemen steht das DMPS ebenfalls über REST APIs in Interaktion. Die Anforderungen zur Steuerung der Prozessinstanzen im System sowie die Nutzung deren Funktionalitäten bieten vier REST Methoden, deren Nutzung nur die Kenntnis über eine einzige Instanz-ID voraussetzt. Zum Zweck der Mitteilung dieser IDs gegenüber dem HCIS, sowie prozessbezogener Informationen und Prozess-Fehler werden generisch zwei APIs beschrieben, die vom HCIS implementiert werden müssen. Weiterhin werden in dieser Arbeit nicht-funktionale Anforderungen identifiziert, die sich grundsätzlich in drei Bereiche aufteilen lassen: die Offenheit, die Erweiterbarkeit und die Zuverlässigkeit des Systems.

Die Interoperabilität und die einfache Integration als Offenheit-Eigenschaften des Systems werden durch den Einsatz des REST-Architekturstils beachtet und sichergestellt. So wurden mit REST einerseits uniforme und minimale Schnittstellen spezifiziert. Andererseits bieten die auf HTTP aufbauende RESTful Webservices eine relativ einfache Integration, da sie über dasselbe HTTP-Protokoll kommunizieren und nicht über eigene anwendungsspezifische Protokolle (z.B. das SOAP-Protokoll). Darüber hinaus entfällt bei der Integration der Plattform hinsichtlich der generisch beschriebenen HCIS APIs der Aufwand für die Spezifikation eines geeigneten Protokolls, da das HTTP-Protokoll bereits vorgegeben ist. Als Folge dieser Ausführungen kann die Logik hinter der generischen HCIS APIs in jeder beliebigen Programmiersprache implementiert werden. Die Workflow-Engine als Kernbaustein der Anwendung bietet ebenfalls Vorteile bezüglich der einfachen Systemintegration. Da die Engine auf der PVM aufgebaut ist, die als eine einfache Java-Bibliothek in allen Java-Umgebungen ausgeführt werden kann, können alle Komponenten des DMPS nur in einem Servlet-Container betrieben werden. Darüberhinaus ist Dank der Portierbarkeit, des breiten Einsatzspektrums und der großen Verbreitung der Java-Technologie auch die einfache Integration in bestehende Systeme und die einfache Erweiterung gewährleistet. Die einfache Erweiterbarkeit des Systems ist zusätzlich durch die Trennung von Modell und Logik in dem Workflow unterstützt.

Die Zuverlässigkeit des Systems ist grundsätzlich durch jBPM und den Workflow gewährleistet. Zum einen bietet die Persistenzhaltung in einer DerbyDB einen Recovery-Mechanismus im Falle eines Systemfehlers. Zum anderen sieht auch der Workflow Fehlerbehandlungsmechanismen vor.

11 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Ansatz für die Unterstützung von verteilten Behandlungsprozessen vorgestellt und eine Plattform als technische Umsetzung entwickelt. Zur Übertragung von Prozessinformationen und Behandlungshistorien wurden zwei Ansätze evaluiert und eine Art „Prozesszustandsdokument“ instrumentiert, das im Mittelpunkt des Behandlungsprozesses als Informationsobjekt steht. Weiterhin wurde ein erweitertes Modell des klassischen diagnostisch-therapeutischen Zyklus in der Medizin zur Unterstützung der extern ablaufenden Behandlungsprozesse eingeführt. Das Modell wurde dann als Ausgangspunkt bei der Modellierung und Implementierung des DMPS-Workflows mit jBPM verwendet. Dieser Workflow bildet die Basis der entwickelten Plattform. Für die externen Interaktionen wurden mit dem REST-Architekturstil minimale und interoperable Schnittstellen zur Verfügung gestellt.
Das DMPS ermöglicht eine bidirektionale Kommunikation mit DMP-Systemen, bietet aber für non-DMPS-Benutzer nur die Möglichkeit durch das Webportal Dokumente an das System zu übergeben. Es entsteht eine Kommunikationslücke, da der Austausch nur in Richtung des DMPS und nicht zurück erfolgen kann. Ein weiteres Ziel bezüglich einer Erweiterung der Plattform könnte somit die Bewerkstelligung einer Rückkopplung mit non-DMPS-Benutzer über „Simple Mail Transfer Protocol“ (SMTP) sein. Notwendig wäre aus Sicht der DMPS-Anwendungsentwicklung zumindest noch die folgende Unterstützung:

- Entwicklung und Integration einer Komponente zur Erstellung von MIME-Nachrichten und Übertragung über SMTP.
- Workflow-Modell-Erweiterung zur Unterstützung von SMTP als Rückkopplung bei verteilten Behandlungsabläufe
- Erweiterung des Webportals zur Ermöglichung der Eingabe von E-Mail Informationen.
- Ein Verfahren zur sicheren Übertragung der Nachrichten (PGP, S/MIME)

Speziell für den Bereich des Gesundheitswesens sollen moderne sicherheitstechnische Methoden und Mechanismen bezüglich der medizinischen Informationen genutzt werden. Da diese Anforderung nicht als Ziel in dieser Arbeit vorgesehen wurde, ist die Nutzerauthentifikation und die Sicherheit beim Transport somit eine weitere wichtige Zielsetzung. Bei der Verwendung von HTTP als eine REST Implementierung für den Austausch von patientenbezogenen Informationen ergeben sich strenge Anforderungen an die IT-Sicherheit. Im Hinblick auf HTTP existieren für die großen Problemfelder der Nutzerauthentifikation und Sicherheit bereits erprobte Lösungen. Dies sind unter anderem spezielle Verfahren zur Authentifikation über Zertifikate oder auch Verschlüsselung z.B. durch HTTP over TLS.

A ANHANG

A.1 Das Prozesszustandsdokument

Das XSD-Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="CDAextension">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="globalProcessInstanceID"
                type="xs:string"/>
                <xs:element name="ResponseEndpoint" type="xs:anyURI"/>
                <xs:element ref="treatmentParticipantsHistory"
                minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="treatmentParticipantsHistory">
        <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="representedOrganization"
                minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="treatmentProcessFork" minOccurs="0"
                maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="representedOrganization">
        <xs:complexType>
            <xs:attribute name="ID" type="xs:integer" use="required"/>
            <xs:attribute name="infoURL" type="xs:anyURI"
            use="optional"/>
            <xs:attribute name="Timestamp" type="xs:long"
            use="optional"/>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="treatmentProcessFork">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="representedOrganization"
                minOccurs="1" maxOccurs="unbounded"/>
                <xs:element ref="treatmentProcessFork" minOccurs="0"
                maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Prozesszustandsdokumente als Beispiel einer sternförmigen und kettenförmigen Behandlung

```
<?xml version="1.0" encoding="UTF-8"?>
<CDAextension xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="CDAextension.xsd">
    <globalProcessInstanceID>null</globalProcessInstanceID>
    <ResponseEndpoint>null</ResponseEndpoint>
    <treatmentParticipantsHistory>
        <representedOrganization ID="1" Timestamp="1231940275828"
        infoURL="http://localhost:8081/RESTServices/info.jsp"/>
        <treatmentProcessFork>
            <representedOrganization ID="2" Timestamp="0"
            infoURL="http://localhost:8082/RESTServices/info.jsp"/>
        </treatmentProcessFork>
        <treatmentProcessFork>
            <representedOrganization ID="3" Timestamp="0"
            infoURL="http://localhost:8083/RESTServices/info.jsp"/>
        </treatmentProcessFork>
    </treatmentParticipantsHistory>
</CDAextension>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<CDAextension xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="CDAextension.xsd">
    <globalProcessInstanceID>null</globalProcessInstanceID>
    <ResponseEndpoint>null</ResponseEndpoint>
    <treatmentParticipantsHistory>
        <representedOrganization ID="1" Timestamp="1231940275828"
```

```
        infoURL="http://localhost:8081/RESTServices/info.jsp"/>
        <representedOrganization ID="2" Timestamp="0"
        infoURL="http://localhost:8082/RESTServices/info.jsp"/>
        <representedOrganization ID="3" Timestamp="0"
        infoURL="http://localhost:8083/RESTServices/info.jsp"/>
    </treatmentParticipantsHistory>
</CDAextension>
```

A.2 Die HCIS APIs

```
@Path("/ExternalDocumentsNotification ")
@POST
@ConsumeMime({"multipart/form-data", "application/xml",
"multipart/mixed"})
@ProduceMime("text/html")
public Response RequestListener (@Context HttpServletRequest request)

public class RequestListener extends HttpServlet {
  protected void processRequest(HttpServletRequest request,
  HttpServletResponse  response);
}

@Path ("/ProcessErrorsNotification")
@GET
@ProduceMime("text/html")
public Response ProcessErrorsListener (
      @Context HttpServletRequest request)

public class ProcessErrorsListener extends HttpServlet {
  protected void processRequest(HttpServletRequest request,
  HttpServletResponse  response);
}
```

A.3 DMPS XML-Prozessdefinition

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition  xmlns="urn:jbpm.org:jpdl-3.2"
    name="DMPModellOrg1">
    <start-state name="patient-related information received">
        <transition to="state information decision">
        </transition>
    </start-state>
    <node name="document with valid ID">
        <event type="node-enter">
            <action class="org.promed.actions.ResumeExistingInstance"
            name="ResumeExistingInstance">
            </action>
        </event>
        <transition to="treatment ready" name="to treatment ready">
        </transition>
    </node>
    <decision name="state information decision">
        <handler class="org.promed.handlers.DecisionMakerExistsId">
        </handler>
        <transition to="document with valid ID" name="existing">
        </transition>
        <transition to="decision making" name="not existing">
            <action name="SendKISNotification"
            class="org.promed.actions.SendKISPostNotification">
            </action>
        </transition>
    </decision>
    <state name="treatment running">
        <event type="node-leave">
            <action name="DocumentsReceived"
            class="org.promed.actions.DocumentsReceived">
            </action>
        </event>
        <transition to="documents received decision" name="are all
        documents received ?">
        </transition>
    </state>
    <state name="send documents">
```

```
    <event type="node-enter">
        <action class="org.promed.actions.SendDocuments"
        name="SendDocuments">
        </action>
    </event>
    <transition to="all documents sent decision" name="are all
    documents sent ?">
    </transition>
    <transition to="notify HCIS" name="ResponseErrorReceived">
    </transition>
</state>
<node name="extern treatment initiation">
    <event type="node-enter">
        <action
        class="org.promed.actions.PrepareTreatmentInitiation"
        name="PrepareTreatmentInitiation">
        </action>
    </event>
    <transition to="send documents" name="to send documents">
    </transition>
</node>
<state name="decision making">
    <description>
    </description>
    <transition to="extern treatment initiation" name="to extern
    treatment initiation">
    </transition>
    <transition to="return documents" name="to return documents">
    </transition>
</state>
<node name="excepiton caught">
    <transition to="treatment ready">
    </transition>
</node>
<state name="return documents">
    <event type="node-enter">
        <action name="SendBackDocuments"
        class="org.promed.actions.SendBackDocuments">
        </action>
    </event>
```

```
    <transition to="treatment ready" name="to treatment ready">
    </transition>
    <transition to="decision making" name="ResponseErrorReceived">
        <action name="SendErrorNotification"
        class="org.promed.actions.SendKISErrorNotification">
        </action>
    </transition>
</state>
<decision name="all documents sent decision">
    <handler
    class="org.promed.handlers.DecisionMakerSentDocuments">
    </handler>
    <transition to="treatment running" name="to treatment
    running">
    </transition>
    <transition to="send documents" name="to send documents">
    </transition>
</decision>
<state name="notify HCIS">
    <event type="node-enter">
        <action name="SendErrorNotification"
        class="org.promed.actions.SendKISErrorNotification">
        </action>
    </event>
    <transition to="all documents sent decision">
    </transition>
</state>
<decision name="documents received decision">
    <handler
    class="org.promed.handlers.DecisionMakerReceivedDocuments">
    </handler>
    <transition to="decision making" name="to decision making">
        <condition></condition>
        <action class="org.promed.actions.SetOldProcessValues"
        name="SetOldValues">
        </action>
        <action name="SendKISNotification"
        class="org.promed.actions.SendKISPostNotification">
        </action>
    </transition>
```

```
        <transition to="treatment running" name="to treatment
        running">
        </transition>
    </decision>
    <end-state name="treatment ready">
        <event type="node-enter">
            <action name="EndTreatment"
            class="org.promed.actions.EndTreatment">
            </action>
        </event>
    </end-state>
    <action class="org.promed.handlers.ProcessException"
    name="ProcessExceptionAction">
    </action>
</process-definition>
```

A.4 UML-Klassendiagramme

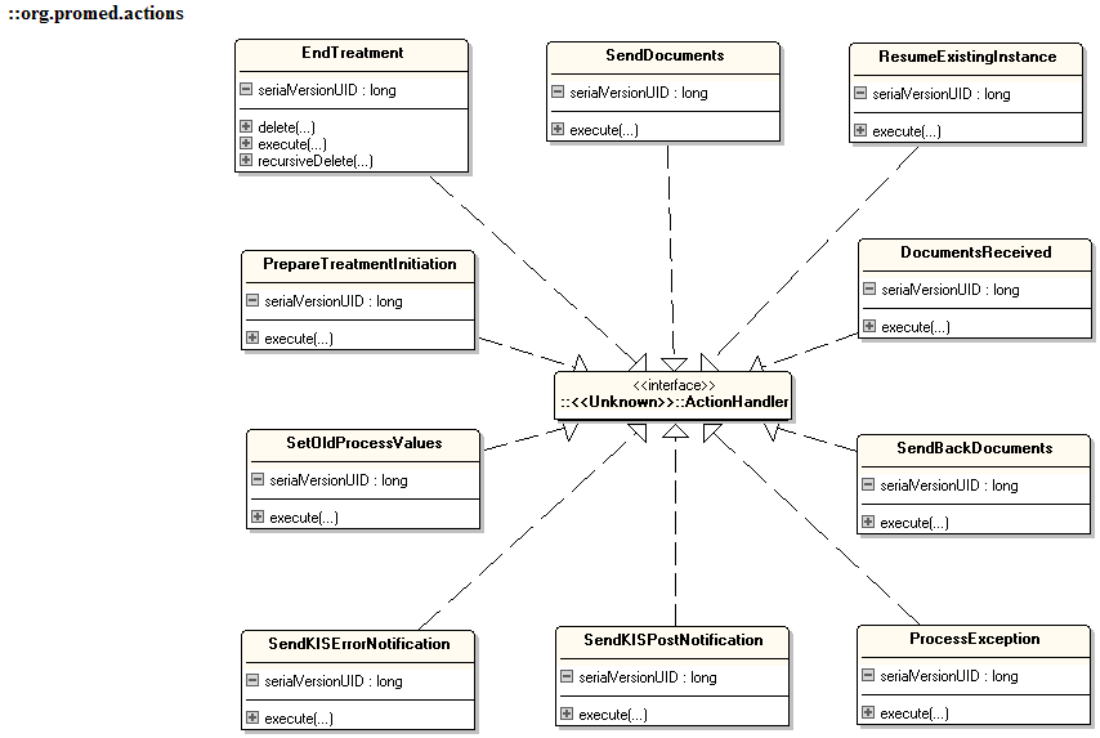


Abbildung 34: Das org.promed.actions Paket

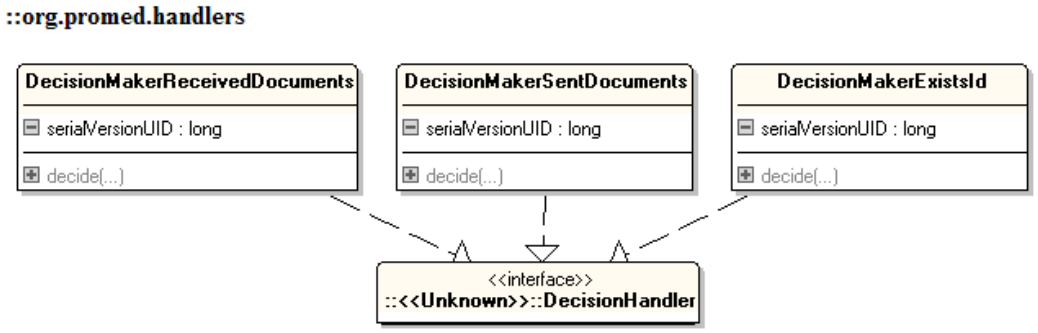


Abbildung 35: Das org.promed.handlers Paket

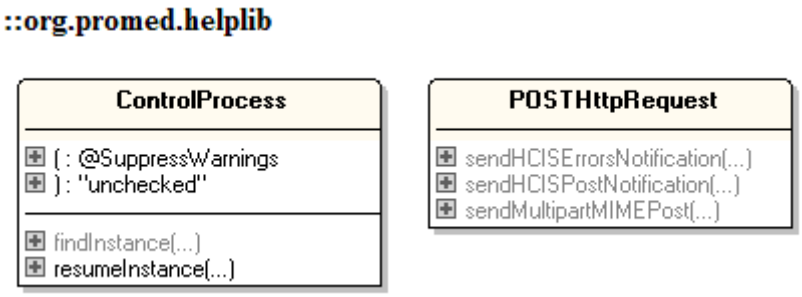


Abbildung 36: Das org.promed.helplib Paket

Literaturverzeichnis

[Bayer02] Bayer T.: „REST Web Services. Eine Einführung.“, 2002. http://www.oio.de/public/xml/rest-webservices.pdf

[Dadam00] Dadam P., Reichert M., Kuhn K.: “Clinical Workflows - The Killer Application for Process-oriented Information Systems? “. In: Abramowicz, W.; Orlowska, M.E. (Eds.): BIS 2000 - Proc. of the 4th Int'l Conference on Business Information Systems, Poznan, Poland, April 2000, Springer-Verlag, 2000, S. 36-59. http://www.informatik.uni-ulm.de/dbis/01/dbis/downloads/DRK00.pdf.

[Fielding00] Fielding T. R.: “Architectural Styles and the Design of Network-based Software Architectures“, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[Flemming08] Flemming D., Giehoff C., Hübner U.: „Entwicklung eines Standards für den elektronischen Pflegebericht auf Basis der HL7CDA Release 2“, GMDS Stuttgart, 15.-19.09.2008. German Medical Science GMS Publishing House, 2008.

[Hinchley07] Hinchley A.: “Understanding Version 3.” Alexander Mönch Publishing, 2007, ISBN 3-933819-21-0.

[HL7CH] HL7 Benutzergruppe Schweitz: „CDA-CH: Spezifikation zum elektronischen Austausch von medizinischen Dokumenten in der Schweiz. Basierend auf der HL7 Clinical Document Architecture (CDA), Release 2.“. http://www.hl7.ch/default.asp?tab=2&item=standard.

[Jablonski97] Jablonski S.: “Workflow-Management-Entwicklung von Anwendungen und Systemen”, Dpunkt-Verlag 1997, ISBN 3-920993-73-X.

[JAXRS] JSR 311: JAX-RS: “The Java™ API for RESTful Web Services”. http://jcp.org/en/jsr/detail?id=311.

[jBPM09a] jBPM, JIRA. https://jira.jboss.org/jira/browse/jbpm.

[JBPM09b] JBoss jBPM website. www.jboss.com/products/jbpm.

[jBPM09c] jBPM jPDL User Guide. http://docs.jboss.com/jbpm/v3.2/userguide/html/.

[Jersey] RESTfulWeb Services Jersey Developer's Guide, http://dlc.sun.com/pdf/820-4867/820-4867.pdf.

[Kaltenborn99] Kaltenborn K.: „Informationstransfer und Wissenstransfer in der Medizin und im Gesundheitswesen.“ Klostermann Verlag 1999, ISBN-10 3465029488.

[Leape95] Leape L. L., Bates D. W., Cullen D. J., Cooper J., Demonaco H. J., Gallivan T., Hallisey R., Ives J., Laird N., Laffel G. and al. et Department of Health Policy and Management, Harvard School of Public Health, Boston: „Systems analysis of adverse drug events ADE prevention study Group JAMA“, 1995; 274: 75-6.

[Lu99] Lu G.: “Multimedia Database Management Systems.” Artech House, 1999, ISBN 0-89003-432-7.

[Manageability] Blog zu Opensource-Workflow-Engines in Java. http://www.manageability.org/blog/stuff/workflow_in_java/view

[Müller07] Müller S.: „Modellbasierte IT-Unterstützung von wissensintensiven Prozessen“, 2007 Dissertation.

[Pflüglmayer01] Pflüglmayer M.:„Informations- und Kommunikationstechnologien zur Qualitätsverbesserung im Krankenhaus.“, 2001 Dissertation.

[POSIX92] IEEE: Portable Operating System Interface for Computer Environments (POSIX). IEEE Standard 1003.0: “Guide to POSIX open Systems Environment”, 1992.

[Prokosch08] Prokosch, H. U.: “Auf dem Weg zur elektronische Krankenakte”, 2008. http://ddi.informatik.uni-erlangen.de/Service/veranstaltungen/schnuppi/2008/Folien_Prokosch.pdf?language=de.

[WfMC09] Workflow Management Coalition: „Workflow Reference Model“. www.wfmc.org/standards/referencemodel.htm.

[Wohed07] Wohed P., Andersson B., Hofstede A., Russell N., van der Aalst W.: „Patterns-based Evaluation of Open Source BPM Systems: The Cases of jBPM, OpenWFE, and Enhydra Shark“, 2007 http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2007/BPM-07-12.pdf.