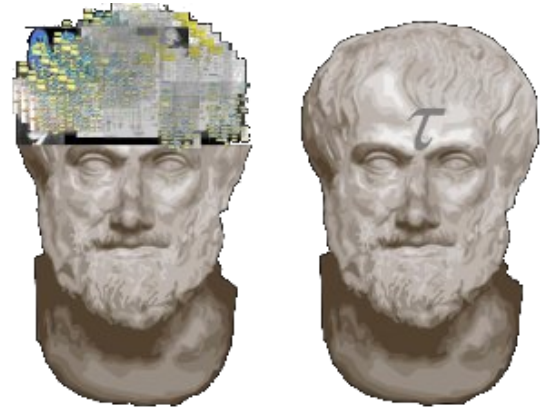


■ fakultät für informatik

PG DynamicFlow

PG DynamicFlow Endbericht

27. April 2009



i n t e r n e b e r i c h t e
i n t e r n a l r e p o r t s

Lehrstuhl XIV
Fakultät für Informatik
Technische Universität Dortmund

Veranstalter

Prof. Dr. Jakob Rehof
Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl XIV
(Geschoßbau IV)
Baroper Straße 301
44227 Dortmund

Betreuer

- Lehrstuhl XIV
Dipl.-Inform. Markus Doedt
Dipl.-Inform. Martin Sugioarto
- Fraunhofer ISST
Dipl.-Inform. Jan Neuhaus
Dipl.-Inform. Claudia Reuter
Dr. Markus Wiedeler

Teilnehmer

Amine Bettaieb
Tobias Böcker
Sadok Chaouech
Tobias Kubissa
Jens Neuhalfen
Manh Linh Nguyen
Stephan Rudolph
Ji Shen
André Smolarczyk
Tobias Sudhölter
Oleksiy Svyerdyelov
Michael Tewes

Kontakt

pg-dynamicflow@lists.cs.uni-dortmund.de

Zeitraum

Sommersemester 2008 und Wintersemester 2008/2009

Inhaltsverzeichnis

1	DynamicFlow Ziele und Aufgaben	1
1.1	Ziel des Projekts	1
1.2	Aufteilung der Aufgaben	2
1.3	Aufbau der Arbeit	3
2	Motivation	5
2.1	Klinikalltag	5
2.2	Medizinische Leitlinien	6
2.3	Klinische Pfade	9
2.4	Klinische Informationssysteme	11
2.5	Workflowmanagement	12
3	Lösungsansatz	17
3.1	Designziele	17
3.1.1	Identifizierung und Modellierung von Prozessen	17
3.1.2	Funktionale Ziele	18
3.1.3	Begründung	20
3.1.4	Ergonomie-Anforderungen	22
3.2	Nutzbarkeit existierender Ansätze	23
3.2.1	Petri-Netze	23
3.2.2	Ereignisgesteuerte Prozessketten	28
3.2.3	ADEPT	30
3.2.4	AgentWork	31
3.2.5	Controlflow-Constraints	32
3.2.6	Dataflow-Constraints	36
3.2.7	Workflow-Patterns nach v.d. Aalst	40
3.2.8	Umsetzung klinischer Pfade (Meiler)	44
3.3	Resultierendes Konzept	48
3.3.1	Grundsätzliches Konzept	49
3.3.1.1	Akteure	49
3.3.1.2	Datenmodellierung	51
3.3.2	Instruktionen	54
3.3.3	Instruktionsgruppen	54
3.3.4	Regeln	55
3.3.4.1	Basisregeln	55
3.3.4.2	Ablaufregeln	56
3.3.4.3	Überwachungsregeln	57

3.3.5	Aktionspläne	58
3.3.6	Automatische Entscheider	59
3.3.7	Manueller Entscheider	60
3.3.8	Workflow	61
3.3.8.1	Synchronisationspunkte	62
3.3.8.2	Aufbauregeln	62
3.3.8.3	Varianten	65
3.3.9	Ausführungssemantik der Elemente	65
3.3.10	Korrektheitskriterien	70
3.4	Beschreibung des erhobenen klinischen Pfades	74
3.4.1	Umsetzung des Pfades im Rahmen der PG	75
4	Technologie	82
4.1	Ausführungssprache BPEL	82
4.1.1	Geschichte	82
4.1.2	BPEL vs. BPML	83
4.1.3	BPEL Konzepte	83
4.1.3.1	Einführung in BPEL	83
4.1.3.2	Hauptmerkmale von WS-BPEL 2.0	84
4.2	BPEL4People	90
4.2.1	BPEL4People Einsatzszenarien	90
4.2.2	Eigenschaften von BPEL4People	92
4.3	Transformation nach BPEL	92
4.3.1	Workflows	93
4.3.2	Aktionspläne	94
4.3.3	Ad-hoc-Modifikationen	96
4.3.4	Varianten	96
4.3.5	BPEL Umsetzung eines 1-out-of-M-Join	97
5	Produktauswahl	101
5.1	Editorauswahl	101
5.1.1	Oracle BPEL Designer	101
5.1.2	ActiveBPEL Designer	102
5.1.3	Eclipse BPEL Designer	102
5.1.4	Eclipse BPMN Modeler	103
5.1.5	Intalio BPMN Designer	104
5.1.6	AgilPro	105
5.1.7	Fazit	105
5.2	Engineauswahl	106
5.2.1	Intalio BPMS Server	106
5.2.2	ActiveBPEL	107
5.2.3	Fazit	108

6	Umsetzung	109
6.1	Engine	110
6.1.1	Wesentlicher Umgang mit ActiveBPEL	110
6.1.2	Deployment eines BPEL-Prozesses	115
6.1.3	Deploymentarchiv	117
6.1.4	Architektur ActiveBPEL	124
6.2	Modellierung durch Designer	125
6.2.1	Perspektiven	126
6.2.1.1	Editor	126
6.2.1.2	Repository	131
6.3	Repository	133
6.3.1	Überblick Architektur	133
6.3.2	Repository als Container	136
6.3.3	Erweiterung von Modellen	138
6.3.4	Umsetzung	142
6.4	Interaktion zwischen Engine und Designer	142
6.4.1	Deployment	143
6.4.2	Monitoring	144
6.4.3	Modifikationen	147
6.4.4	Suspend/Resume	147
6.5	Durchführung einer Ad-hoc-Modifikation	150
6.5.1	Namespace DfEngineExtensions als BPEL Extension	152
6.5.2	Präambel Ad-hoc-Deployment	155
6.5.3	Ad-hoc-Deployment	156
6.5.4	Prozessinstanzmigration	158
7	Lessons Learned	163
7.1	Technische Probleme	163
7.2	Einstieg in ActiveBPEL mit AspectJ	166
7.3	Organisatorisch	169
8	Fazit und Ausblick	170
8.1	Bewertung Aktueller Stand	170
8.2	Ausblick und Konzeptionelle Erweiterungen	172
8.3	Danksagung	174
A	Beschreibung des Beispielworkflows	183
A.1	Beschreibung der Aktionspläne	183
A.1.1	AP01: Anamnese	183
A.1.2	AP02: Telefonische Rücksprache	184
A.1.3	AP03: Indikationsbewertung	185
A.1.4	AP04: Vorbereitung der Röntgenuntersuchung	185
A.1.5	AP05: Anamnesebogen	186

A.1.6	AP06: Radiologische Untersuchung Postphase	186
A.1.7	AP07: Letzte Indikationsstellung	187
A.2	Beschreibung der Instruktionen	188
A.3	Liste der Variablen	191

Abbildungsverzeichnis

1.1	Arbeitsaufbau der PG	3
2.1	Evidenzbasierte Medizin.	6
2.2	Idee medizinischer Leitlinien	8
2.3	Aspekte klinischer Pfade: Evidenzbasierung .[89, S. 52]	10
2.4	Aspekte Klinischer Pfade: Organisatorisch .[89, S. 76]	11
2.5	Komponenten der elektronischen Krankenakte (Vgl. MCS-Phoenix [83]).	13
3.1	Kantentypen in ADEPT [92].	30
3.2	Knotenausgänge in ADEPT [92].	30
3.3	Integration von ECA in WFMS [85, S. 5]	32
3.4	Expliziter Datenfluss [58].	39
3.5	Impliziter Datenfluss über den Kontrollfluss [58].	39
3.6	Impliziter Datenfluss über ein „Repository“ [58].	40
3.7	Aspekte Klinischer Pfade [23, S. 42]	45
3.8	Modellhierarchie [23, S. 50]	46
3.9	Aufbauschema eines Aktionsplans.	50
3.10	Interaktion der Nutzergruppen.	51
3.11	Beispiel für einen Regelbaustein im Editor.	56
3.12	Beispiel eines Deadlocks in Petrinetzen. [102, S. 272]	71
3.13	Beispielprozess	71
3.14	Workflow zur Leistung LA03.	75
3.15	Erste Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	76
3.16	Zweite Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	76
3.17	Dritte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	77
3.18	Vierte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	77
3.19	Fünfte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	78
3.20	Sechste Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	78
3.21	Siebte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].	79
3.22	Prozedurbeschreibung zur Leistung LA05 Arztbrief [66].	79
4.1	Entwicklungsfortschritt: von XLANG bis WS-BPEL 2.0	83
4.2	Fachansicht Prozess	93
4.3	BPEL-Ansicht Aktionsplan	94
4.4	Behandlung eines Herzstillstandes	98
4.5	Structured discriminator pattern (Quelle: [22])	98
4.6	Blocking discriminator pattern (Quelle: [22])	99

4.7	Cancelling discriminator pattern (Quelle: [22])	100
5.1	Die Modellierungsschicht des BPMN Modeler.	104
6.1	Architektur	109
6.2	Active Processes Queue	112
6.3	Zustände in ActiveBPEL	113
6.4	Zuweisung von Tasks an Inboxen verschiedener qualifizierter User [24].	114
6.5	Prozesslebenszyklus [79]	116
6.6	Das BPR-Archiv von my_process.bpel	117
6.7	BPRBuilder	121
6.8	Grundlegende Architektur von ActiveBPEL . [105, S. 36]	125
6.9	Liste von auf dem Server installierten Prozessen	127
6.10	Designstudie: Die Darstellung einer Prozessinstanz in der „Ad-hoc-Sicht“.	128
6.11	Designstudie: Ad-hoc-Modifikation: Zwei Varianten werden hinzugefügt.	129
6.12	Designstudie: Die Synchronisation den Aktionsplänen durch Synchronisator.	130
6.13	Designstudie: Die Darstellung eines Aktionsplanes im Editor.	131
6.14	Designstudie: Das Editieren eines Aktionsplanes im Editor.	132
6.15	BPEL-Elemente in der Palette des Designers	133
6.16	Darstellung eines BPEL-Prozesses im Designer	134
6.17	Quelltext eines BPEL-Prozesses im Editor	134
6.18	Die BPEL-Sicht des Designers.	135
6.19	Das Beispiel eines neuen BPEL-Konstruktes	136
6.20	Die Schaltfläche zur Erstellung der BPR-Datei aus einem Projekt	136
6.21	Die Erstellung der BPR-Datei aus einem Projekt	137
6.22	Das <i>bpr-build</i> -Verzeichnis in enem Projekt	137
6.23	Die Quelltextsicht des Designers.	138
6.24	Das Kopieren einer Datei ins Repository	139
6.25	Repository Aufbau	140
6.26	Schema einer Repository Organisation	141
6.27	Modell- und Datenformattransformation mit Eclipse Modeling Framework	141
6.28	Query Mechanismus des Repositories	142
6.29	Beispiel eines neu importierten EMF Modells	143
6.30	Repository View - Technische Ansicht	143
6.31	Repository View - Fachliche Ansicht	144
6.32	Zustand einer Prozessinstanz	148
6.33	Zusammenhang Prozessdefinition und BPEL Extensions	153
6.34	Überblick Klassenstruktur für das Ad-hoc-Deployment.	162
8.1	Gruppierte Instruktionen eines Aktionsplanes	174
8.2	Formulare der gruppierten Instruktionen eines Aktionsplanes	175

1 DynamicFlow Ziele und Aufgaben

Die Etablierung von Workflow Management Systemen (WfMS) spielt in vielen industriellen Unternehmungen und im heutigen Dienstleistungsgewerbe eine immer größere Rolle. Ausgehend von der Modellierung der Geschäftsprozesse können WfMS für den optimierten und standardisierten Ablauf der Prozesse sorgen. Die in heutigen betrieblichen Umgebungen zum Einsatz kommenden WfMS sind auf komplexe, vorhersagbare und oft wiederkehrende Prozesse ausgelegt. Dies bedeutet jedoch, dass die Geschäftsprozesse immer gleich aufgebaut sind und alle möglichen Abweichungen im Vorfeld ermittelt, erkannt, definiert und modelliert werden müssen. Zur Laufzeit wird, abhängig von der gegenwärtigen Situation, der passende Pfad ausgewählt und der Prozess entsprechend bearbeitet.

Auch im klinischen Umfeld kommt es heutzutage aufgrund des steigenden Kostendrucks immer mehr darauf an, den Behandlungsprozess rund um den Patienten zu optimieren und zu standardisieren. Hier stellt sich jedoch heraus, dass die medizinische Behandlung jedes Patienten, sowohl in der Diagnostik als auch bei der Therapie, individuelle Lösungswege erfordert. An dieser Stelle geraten klassische WfMS an ihre Grenzen, da sie meist nicht dafür ausgelegt sind, Ad-hoc-Modifikationen an vormodellierten Prozessen vorzunehmen [88]. Aufgrund der starken Diversifikation ist es kaum möglich, alle Alternativen zu modellieren. Würde versucht werden, alle Formen einer Behandlung zu modellieren, wäre zum einen nicht sichergestellt, dass auch alle möglichen Optionen erfasst werden und zum anderen würde ein solches Modell, aufgrund mangelnder Übersichtlichkeit nicht mehr praktisch nutzbar sein.

1.1 Ziel des Projekts

Das Ziel der Projektgruppe (PG) DynamicFlow ist es, ausgehend von klinischen Pfaden, die als Referenzmodell für eine Behandlung stehen, und existierenden Modellierungsformen für Geschäftsprozesse einen Weg zu finden, Behandlungsprozesse adäquat darzustellen. Aus diesen Modellen werden, durch verschiedene Abstraktionsstufen hindurch, dynamisch ausführbare Elemente einer Workflow-Technologie erzeugt. [18]

Die von der PG erstellten Modelle werden von verschiedenen Benutzergruppen, sowohl aus der Informatik als auch aus der Medizin angewendet. Verschiedene, für den jeweiligen Benutzer angepasste Darstellungsformen ermöglichen einen flexiblen Einsatz im Klinikalltag. Durch die konsistente Modellierung und Optimierung der Behandlungsprozesse, entlang der klinischen Pfade können unnötige Kosten, die z.B. durch doppelte diagnostische oder therapeutische Maßnahmen entstehen, vermieden werden. Dies kann die Existenz eines Krankenhauses sichern und sorgt somit für eine umfassende Versorgung des Patienten.

1.2 Aufteilung der Aufgaben

Aufgrund der verschiedenen Themenbereiche bietet sich eine Aufteilung in drei Projektteams mit den folgenden Aufgabenschwerpunkten an:

1. Workflow Templates (Team-A)

Ausgehend von medizinischen Leitlinien und klinischen Pfaden werden mögliche Workflow Templates ermittelt und ein Konzept für deren Umsetzung erstellt. Ein zentraler Aspekt ist die Unterstützung von Ad-hoc-Modifikationen. Das bedeutet, dass es nicht nötig ist, den kompletten Workflow bis ins kleinste Detail im Voraus zu modellieren, sondern dass während der Ausführung einzelne Anpassungen durchgeführt werden können. Des Weiteren werden zu Fallbeispielen mögliche Anwendungsszenarien aufgezeigt, deren Implementierung von den anderen Projekt-Teams umzusetzen ist. Hierzu werden für den jeweiligen Fachanwender angepasste Sichten der Modelle zur Verfügung gestellt.

2. Workflow Modellierung (Team-B)

Die o. g. Templates sollen in der Workflow-Ausführungssprache Business Process Execution Language (BPEL) abgebildet werden. Um dies in adäquater Form lösen zu können, wird zunächst eine Marktanalyse über Modellierungswerkzeuge erstellt. Hierbei erfolgt eine Einschränkung auf Open-Source Lösungen, da es andernfalls höchstwahrscheinlich nicht möglich ist, die Anwendung im Quelltext zu erhalten und anzupassen. Somit wird eine Modellierungsumgebung ausgewählt, die folgende Anforderungen bereits erfüllt oder entsprechend angepasst werden kann:

- **Verschiedene Sichten**

Es muss möglich sein, den modellierten Workflow so darzustellen, dass es für Anwender aus verschiedenen Disziplinen möglich ist, Änderungen im Prozessablauf vorzunehmen. Hierzu soll es eine fachliche und eine technische Ansicht geben.

- **Transformation in BPEL**

Das erstellte Modell muss (automatisch) in ausführbaren BPEL Code transformiert werden können.

- **Human Tasks**

Die einzelnen elementaren Aktivitäten, d.h. von menschlichen Akteuren ausgeführte Arbeitsschritte, sollen als so genannte Human Tasks abgebildet werden können.

- **Datenhaltung**

Alle erstellten Workflows und Ihre einzelnen Komponenten sollen in einem zentralen Datenspeicher (dem Repository) gehalten werden können, so dass sie als Bausteine zur Konstruktion neuer Modelle bzw. Workflows dienen können.

3. Workflow Engine (Team-C)

Ähnlich zur Marktanalyse die von Team-B durchgeführt wird, wählt Team-C eine

Open Source Engine aus, die sowohl in der Lage ist, die erstellten Workflows auszuführen als auch auf Ad-hoc-Modifikationen reagieren kann. Hierzu müssen Signale aus der Kommunikation zwischen Modellierungsanwendung und Ausführungsengine ausgewertet und darauf reagiert werden.

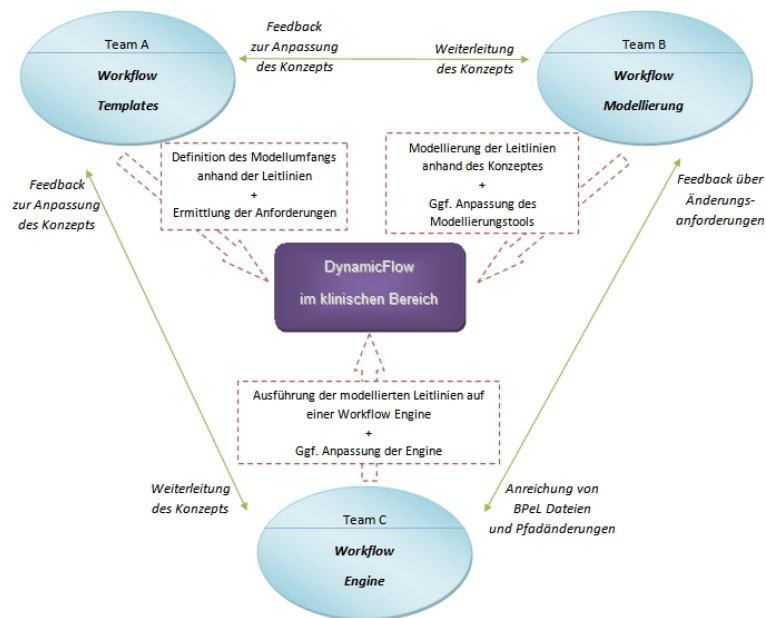


Abbildung 1.1: Arbeitsaufbau der PG .

1.3 Aufbau der Arbeit

Der inhaltliche Aufbau dieser Arbeit orientiert sich in etwa an den Aufgabengebieten der einzelnen Projektteams. Zunächst werden in Kapitel zwei die fachlichen Hintergründe und Vorteile des Einsatzes von Informations- und Workflowmanagementsystemen im Klinikalltag vorgestellt. Anschließend werden im dritten Kapitel zunächst die Grundlagen und Designziele, sowie mögliche bereits etablierte Lösungsansätze vorgestellt. Ferner werden die Vor- und Nachteile der Ansätze diskutiert aus denen schließlich ein neues Konzept zur Umsetzung von Ad-hoc-Workflows im klinischen Umfeld resultiert. Das dritte Kapitel schließt mit der Vorstellung des, von der PG entwickelten Konzeptes und einer beispielhaften Umsetzung eines realen Behandlungspfades ab. Das vierte Kapitel befasst sich mit BPEL, der zugrunde liegenden Technologie und der Transformation der Modelle in BPEL. Das fünfte Kapitel beschreibt das Vorgehen bei den Marktanalysen im Bereich der Modellierungs- und Ausführungsanwendungen. Die Schnittstellen zwischen den einzelnen Bearbeitungsebenen und die Umsetzung der Implementierung, sowie Anpassung der einzelnen Anwendungen wird in Kapitel sechs vorgestellt. Kapitel sieben fasst die Erfahrungen zusammen, die aus technischer und organisatorischer Sicht gemacht wurden und wie die aufgetretenen Probleme

angegangen wurden.

Abschließend sei angemerkt, dass aus Gründen der Lesbarkeit im Text die männliche Form gewählt wurde, nichtsdestoweniger beziehen sich die Angaben auf Angehörige beider Geschlechter.

2 Motivation

Wie einleitend dargestellt soll ein Konzept entwickelt werden, das es auch in Einrichtungen mit dynamischen Prozessabläufen ermöglicht nach vordefinierten Mustern vorzugehen. Im Rahmen der PG wurde der das Gesundheitswesen im Allgemeinen und das Krankenhausumfeld im Besonderen als Anwendungsdomäne festgelegt. Hierbei stellt sich die Herausforderung, den Anwendern aus informatikfernen Disziplinen eine Möglichkeit anzubieten, ad hoc in den vormodellierten Prozess einzugreifen und Anpassungen vorzunehmen. Hierbei muss sichergestellt sein, dass der Mediziner letztlich immer die entscheidende Instanz darstellt, obgleich vom System eine erprobte Vorgehensweise vorgeschlagen wird.

2.1 Klinikalltag

Um Patienten die bestmögliche Behandlung zukommen zu lassen und trotzdem kosteneffizient zu arbeiten, ist es notwendig, dass entlang wohldefinierter Konzepte und Empfehlungen gearbeitet wird. In einigen Häusern wurden bereits zentrale Notaufnahmen eingerichtet, um zu verhindern, dass Patienten vom Aufnahmepersonal in der Ambulanz zum falschen Facharzt weitergeleitet werden [21]. Es kann z.B. vorkommen, dass eine Patientin mit unklaren Unterbauchbeschwerden von einem Pförtner, ohne Beachtung von Details auf die gynäkologische Station überwiesen wird. Nach längerer Wartezeit befasst sich ein Mediziner mit der Patientin. Die Wartezeit entsteht, da die akute Behandlung, im Tagesgeschehen auf der Station, eingeschoben werden muss. Hier wird festgestellt, dass der Anfangsverdacht auf eine gynäkologische Erkrankung nicht zutrifft und die Patientin mit Verdacht auf eine akute Appendizitis in die innere Medizin weitergeleitet wird.

In diesem Szenario werden durch doppelte Maßnahmen, z.B. Anamnese und Diagnostik, unnötig wertvolle Zeit und knappe Finanzmittel verschwendet. Das Konzept der zentralen Notaufnahme sieht vor, dass sich von Anfang an speziell geschultes, medizinisches Personal mit den eintreffenden Patienten befasst und somit die Wahrscheinlichkeit eine korrekte Erstdiagnose zu erstellen stark erhöht wird. Durch eine zentrale Datenhaltung und ein Informationssystem, das die einmal erhobenen Daten an die Personen weiterleitet, die den nächsten Bearbeitungsschritt durchführen, können derartige Situationen vermieden werden.

Hinzu kommt, dass Klinikpersonal grundsätzlich eine knappe Ressource darstellt und je weniger die Mediziner oder das Pflegepersonal mit administrativen und organisatorischen Tätigkeiten belastet werden, desto mehr sind sie in der Lage, Patienten umfassend zu behandeln. Viele Mediziner berichten gerade davon, dass der Aspekt der medizinischen Arbeit immer mehr in Verwaltungstätigkeiten untergeht. Dies schränkt letztlich die Zufriedenheit

des Krankenhauspersonals bei der Erfüllung ihrer alltäglichen Aufgaben und somit auch die Behandlungsqualität ein.

2.2 Medizinische Leitlinien

Dieser Abschnitt zeigt einen Weg auf, wie im Klinikalltag dafür gesorgt werden kann, dass je nach Diagnose eine adäquate Behandlung durchgeführt wird, die sich an gewissen Standards, den medizinischen Leitlinien, orientiert. Um eine Einführung in medizinische Leitlinien zu geben, soll hier zunächst der Begriff der evidenzbasierten Medizin (EbM) eingeführt werden.

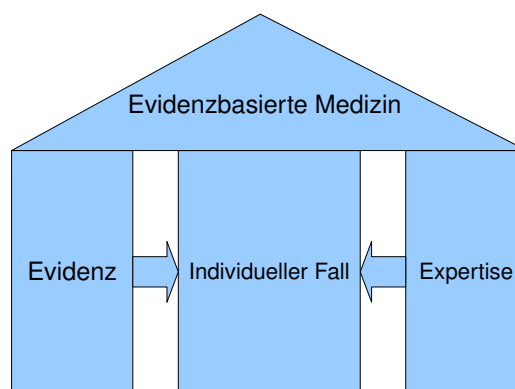


Abbildung 2.1: Evidenzbasierte Medizin.

„EbM ist der gewissenhafte, ausdrückliche und vernünftige Gebrauch der gegenwärtig besten externen, wissenschaftlichen Evidenz für Entscheidungen in der medizinischen Versorgung individueller Patienten. Die Praxis der EBM bedeutet die Integration individueller klinischer Expertise mit der bestmöglichen externen Evidenz aus systematischer Forschung.“^[44]

Evidenz bezeichnet dabei nicht *nur* wissenschaftlich fundiertes Wissen (externe Evidenz), sondern integriert auch das persönliche Erfahrungs- und Expertenwissen des Arztes, die klinische Expertise. Abbildung 2.1 veranschaulicht das Zusammenspiel der verschiedenen Aspekte bei der Behandlung eines individuellen Patienten. Hierbei ist die EbM gewissermaßen die Kombination aus wissenschaftlicher Evidenz und der bis dato als praktisch bestmöglich angesehenen Behandlungsweise.

Erste Ansätze evidenzbasierter Medizin gab es Mitte des 18. Jahrhunderts in Britanien. Der Begriff selbst wird erstmals Ende des 18. Jahrhunderts von dem schottischen Arzt George Fordyce in seinem Aufsatz: „An attempt to improve the Evidence of Medicine“ verwendet. Evidenzbasierte Medizin ist also der Versuch medizinisches Handeln zu systematisieren und

ein Werkzeug zur Evaluierung zu schaffen.

Ab 1972 wurden die Bemühungen evidenzbasierte Medizin voran zu bringen intensiviert [96]. Seit 1994 wird die Erstellung von Leitlinien vom „Sachverständigenrat für die konzer-tierte Aktion im Gesundheitswesen“ empfohlen. Die medizinischen Fachgesellschaften sind danach aufgefordert die Leitlinien zu erstellen [61, S. 6f].

Die Arbeitsgemeinschaft der „Wissenschaftlichen Medizinischen Fachgesellschaften e.V.“ (AWMF) definiert medizinische Leitlinien, wie folgt:

„Leitlinien sind systematisch entwickelte Darstellungen und Empfehlungen mit dem Zweck, Ärzte und Patienten bei der Entscheidung über angemessene Maßnahmen der Krankenversorgung (Prävention, Diagnostik, Therapie und Nachsorge) unter spezifischen medizinischen Umständen zu unterstützen.“ [27]

Die AWMF liefert noch eine zusätzliche, enger gefasste Definition für Leitlinien:

„Die Leitlinien der AWMF sind systematisch entwickelte Hilfen für Ärzte zur Entscheidungsfindung in spezifischen Situationen. Sie beruhen auf aktuellen wissenschaftlichen Erkenntnissen und in der Praxis bewährten Verfahren und sorgen für mehr Sicherheit in der Medizin, sollen aber auch ökonomische Aspekte berücksichtigen. Die Leitlinien sind für Ärzte rechtlich nicht bindend und haben daher weder haftungs begründende noch haftungsbefreiende Wirkung.“ [28]

Hieraus folgt, dass Leitlinien keine reinen Beschreibungen von Handlungsabläufen oder gar Entscheidungsgraphen darstellen. Leitlinien stellen demnach einen zusammenfassenden Bericht zum aktuellen Stand der Forschung, ein bestimmtes Krankheitsbild betreffend dar. Des Weiteren enthalten Leitlinien eine Diskussion und empfehlen bestimmte Behandlungsmethoden, deren antizipierter Nutzen innerhalb der medizinischen Fachabteilungen, interdisziplinär, sowie nach außen mit Leistungsträgern und bestehenden Richtlinien abgeglichen ist.

Abbildung 2.2 stellt die Entwicklung von medizinischen Leitlinien dar. Hierbei handelt es sich um ein Dokument, in dem das theoretische medizinische Wissen und die vorhandene Expertise aus verschiedenen Fachrichtungen unter Berücksichtigung von Qualitätsstandards zusammengestellt sind. Dieses gibt dem Mediziner einen Handlungsvorschlag zu einer gesundheitlich optimalen, kostengünstigen und insgesamt anerkannten Behandlung. Im Idealfall steht dem Mediziner bei einer bestimmten therapeutischen Maßnahme ein evidenzbasiertes, medizinisches Basiswissen zur Verfügung. Die Durchsicht der, meist sehr umfangreichen, Literatur zu einer Diagnose und das Abwägen aller Gesichtspunkte ist somit nur noch im Bedarfsfall notwendig. Allerdings ist die rechtliche Lage in Bezug auf medizinische Leitlinien noch nicht endgültig geklärt:

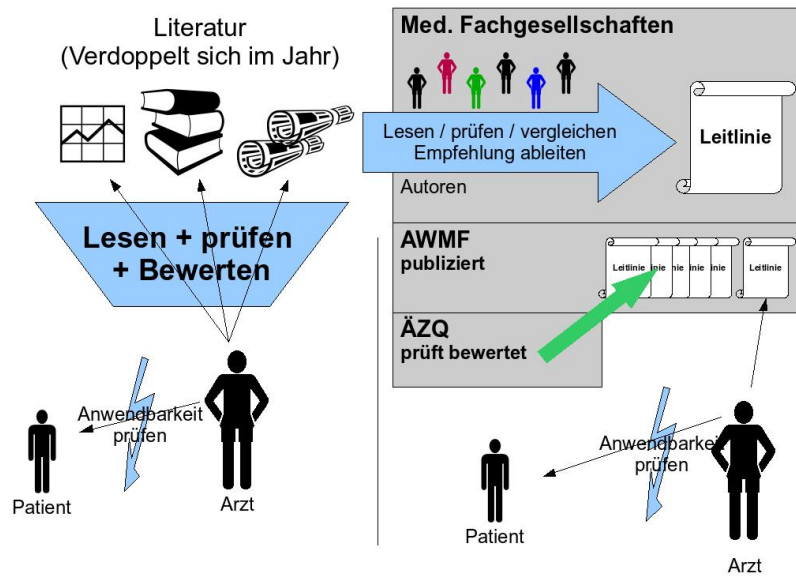


Abbildung 2.2: Idee medizinischer Leitlinien .

„Richtlinien sind Handlungsregeln einer gesetzlich, berufsrechtlich, standesrechtlich oder satzungsrechtlich legitimierten Institution, die für den Rechtsraum dieser Institution verbindlich sind und deren Nichtbeachtung definierte Sanktionen nach sich ziehen kann.“[27]

„Die Leitlinie ist medizinisch verbindlich, wenn sie dem Standard entspricht, und ist rechtlich verbindlich, weil sie dem Standard entspricht.“[64, S. 115]

Es obliegt somit immer noch dem behandelnden Mediziner, die richtige Entscheidung zu treffen. Ist eine Leitlinie aktuell und kann sie als Standard angesehen werden und der Arzt handelt entgegen ihrer Aussage, kann dies als Kunstfehler interpretiert werden. Wird jedoch nach einer nicht standardkonformen Leitlinie gehandelt, so hat dies keine rechtliche Bedeutung. Rechtliche Bedeutung haben Leitlinien zur Zeit nur insofern, als diese in die Berufsverordnung oder in Gesetze eingebunden werden [64]. Dies könnte durchaus auch einer der Gründe dafür sein, dass der Einsatz medizinischer Leitlinien nicht so stark verbreitet ist, wie es wünschenswert wäre [42].

In der aktuellen Diskussion tauchen folgende Begriffe immer wieder auf: Empfehlung, Clinical-Pathways (bzw. Behandlungspfade), Leitlinie und Standard. Sie stellen (aufsteigend sortiert) jeweils eine höhere Verbindlichkeitsstufe dar [64, S. 81]. Der Standard ist dabei das Maß der ärztlichen Behandlung [71, S. 30], nachdem sich diese (aus rechtlicher Sicht) zu richten hat. Hinzu kommen Richtlinien, Versorgungsleitlinien und Beratungsleitlinien, die schon eindeutig rechtliche Bedeutung und somit Verbindlichkeit erhalten haben [64, S. 81f].

Um medizinische Leitlinien in einer visuellen, objektorientierten Beschreibungssprache zu implementieren, wurde das Guide Line Interchange Format (GLIF) an der Stanford University entwickelt. Mit Hilfe von GLIF können medizinische Leitlinien erstellt, verteilt, aktualisiert und umgesetzt werden. GLIF benutzt dazu eine Reihe von Basisobjekten die durch Vererbung spezialisiert sind. Basisobjekte sind Entscheider und Patientenzustände, Verzweigungspunkte und Synchronisationspunkte, die zu einer Leitlinie kombiniert werden können. Die Leitlinien können wiederum geschachtelt werden [49, 72].

2.3 Klinische Pfade

Die folgenden Zitate zu klinischen Pfaden beleuchten ihre bedeutendsten Aspekte und können als Grundlage einer Definition beitragen [63, S.42ff]:

„Behandlungspfade sind der *regelmäßige Handlungsablauf*, den ein Patient während seines gesamten Krankenhausaufenthaltes durchläuft. Dieser Ablauf wird mit Hilfe von organisatorischen Prozessen, Zeitvorgaben und Behandlungsleitlinien sichergestellt. Behandlungspfade setzen die berufsübergreifende Zusammenarbeit voraus, um reibungslose Abläufe zu gewährleisten und sinnvolle Therapien zu bestimmen. Diese stellen den Behandlungspfad dar. Diese Prozessorientierung hat sowohl das Ziel, Abläufe zu standardisieren und transparent zu machen als auch kostengünstige Leistungen zu einer festgelegten Qualität anzubieten. Behandlungspfade sind somit ein *klinikinterner Standard*.“

„Behandlungspfade sind ein *Werkzeug für Diagnose und Therapie*, in der viele Handlungen aufgezeigt und *dokumentiert* werden. Sie schaffen die Möglichkeit Potentiale bezüglich Marketing, Prozessmanagement, Organisation und Qualität aufzudecken. [...]. Sie stellen den Ist-Ablauf einer Behandlung dar und liefern damit die Datengrundlage für eine anschließende *Prozessoptimierung*.“

„Eine Methode und Bestandteil zur *Transparenzschaffung* und Steuerung des medizinischen Leistungsgeschehens, um Qualitätssicherung und -verbesserung zu erreichen.“

„Standardisierter Behandlungsplan, der bestimmte Untersuchungen und Behandlungen an den einzelnen Verweiltagen für *alle Disziplinen* festlegt.“

Klinische Pfade sind also Berufsgruppenübergreifende Handlungsanweisungen auf der Grundlage Medizinischer Leitlinien. Sie dienen der Koordination und der Evidenzbasierung (aufgrund der Leitlinien auf denen sie beruhen) und der Kostenanalyse und Optimierung. Der Scheinbare Widerspruch zwischen Kostenoptimierung und besserer Behandlung wird hier

durch die Optimierung des Prozesses und die Beschränkung auf medizinisch Sinnvolle Behandlungen aufgehoben.

Klinische Pfade sind, im Gegensatz zu medizinischen Leitlinien, abhängig von der Institution in der sie definiert werden. Sie beschreiben alle Aspekte der Behandlung, die ein Patient mit einer bestimmten Erkrankung in einem bestimmten Krankenhaus durchläuft. Klinische Pfade können unter evidenzbasierten, organisatorischen und ökonomischen Aspekten betrachtet werden (siehe Abbildungen 2.3 und 2.4). Obwohl medizinische Leitlinien für alle Institutionen gleich sind, kann ihre Ausprägung und Implementierung in Form von klinischen Pfaden, je nach Institution abweichen. Dies kann aus mehreren Gründen vorkommen: zum einen kann es sein, dass bestimmte Behandlungsempfehlungen aus der Leitlinie nicht durch das, in der Institution vorhandene, Personal durchgeführt werden können, zum anderen kann das Fehlen von entsprechendem Gerät zur Substitution von Untersuchungen durch andere Methoden führen. Auch die nicht durch die Leitlinie erfassten Handlungen (z. B. Buchungen, Transporte, Pflege, Beschaffung, etc.) können von Institution zu Institution variieren.



Abbildung 2.3: Aspekte klinischer Pfade: Evidenzbasierung .[89, S. 52]

Eine Möglichkeit der Modellierung von klinischen Pfaden ist i>PM, welche in der Diplomarbeit von Claudia Reuter beschrieben wird. Hierbei werden verschiedene Objekte für die Modellierung von Prozessen bereitgestellt. Grafisch werden Prozesse durch Rechtecke und Entscheidungen durch Rauten symbolisiert. Die Ausführungsreihenfolge und der Datenfluss werden durch Pfeile repräsentiert. Darüber hinaus können durch Kreise die logischen Operatoren and, or und xor ausgedrückt werden. Im Kontext von i>PM wird die Modellierung von Datenflüssen und darin die Erzeugung von aggregierten Datencontainern ermöglicht. Hierbei können logisch zusammenhängende Daten zusammengefasst werden, z. B. gehören Angaben zu Nikotin- und Alkoholkonsum zur Gruppe der Risikofaktoren einer Operation. Somit kann i>PM als Werkzeug genutzt werden, um klinische Pfade abzubilden [89].

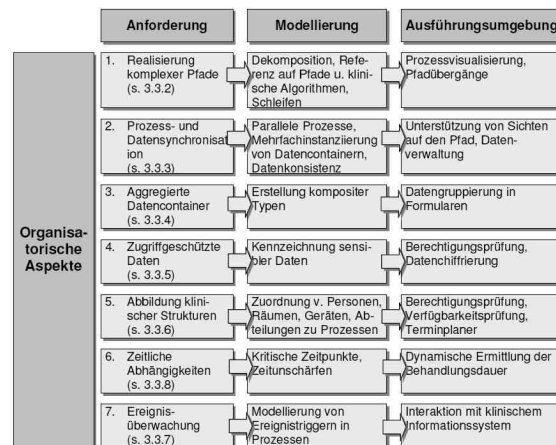


Abbildung 2.4: Aspekte Klinischer Pfade: Organisatorisch [89, S. 76]

Da klinische Pfade als institutionsspezifische Umsetzung medizinischer Leitlinien aufzufassen sind, können auch sie nicht immer strikt eingehalten werden. Folgende Situationen führen dazu, dass vom klinischen Pfad abgewichen werden muss (vgl. [50, S. 21]):

- Unvorhergesehene Ereignisse
- Zustandsänderung des Patienten
- Falsche Therapiesteuerung durch das medizinische Krankenhauspersonal
- Systemprobleme innerhalb des Krankenhauses
- Systemprobleme außerhalb der Einrichtung
- Einflussnahme durch den Patienten
- Einflussnahme sonstiger Bezugspersonen (Eltern, Vormund, etc.) des Patienten

Aufgrund der o.g. Ereignisse eintretenden Abweichungen muss sichergestellt sein, dass auch das WfMS dynamisch auf die veränderten Situationen eingehen kann, d.h. die modellierten Abläufe müssen ad hoc angepasst werden können.

2.4 Klinische Informationssysteme

Ein klinisches Informationssystem (KIS) sorgt für eine umfassende und konsistente, sowie global zugreifbare Datenhaltung innerhalb eines Krankenhauses. Beispielhaft soll hier auf

das KIS MCS-Phoenix eingegangen werden. Hersteller ist ursprünglich die Firma Parametrix Solutions in der Schweiz, der Vertrieb und die Anpassung für den deutschen Markt werden von der MCS-AG übernommen, welche an Parametrix Solutions beteiligt ist. Phoenix ist ein, in Borland Delphi implementiertes, Client-Server-System. Als Server-Betriebssysteme können sowohl Microsoft Windows 2003, als auch Linux eingesetzt werden, als Datenbankserver werden Microsoft SQL-Server 2000 bzw. 2005 und Oracle Database unterstützt. Das Einfügen von Behandlungsdaten in die Datenbank erfolgt formularbasiert, so verfügt jeder Benutzer (Ärzte, Pflege- oder Verwaltungspersonal), der mit dem System in Kontakt kommt über einen speziell auf seine Anforderungen angepassten Zugang zu den relevanten Patientendaten.

Der Zugang erfolgt über die, a priori festgelegte, Zuordnung jedes Benutzers zu bestimmten Benutzergruppen, mit genau spezifizierten Zugriffsrechten und so genannten Einstiegs- punkten. Einstiegs- punkte stellen Übersichten zu Verfügung, wie z. B. alle Patienten einer bestimmten Abteilung oder eines bestimmten Zimmers. Darüber hinaus können virtuelle Einstiegs- punkte definiert werden, wie z. B. "meine Patienten", hier könnte sich der Benutzer eine Übersicht aller, von ihm behandelten Fälle anzeigen lassen.

Zentrales Dokument ist die elektronische Krankenakte (siehe Abbildung 2.5). Diese enthält sowohl die Elemente der klassischen, papiergebundenen Krankenakte, als auch Erweiterungen, die durch die elektronische Abbildung erst ermöglicht werden.

Durch die elektronische Speicherung datenschutzrechtlich relevanten Materials ergeben sich neue Anforderungen an die Krankenakte, wie z. B. Zugriffsschutz oder Historisierung. Ein entscheidender Vorteil der elektronischen Krankenakte ist, dass hier, im Gegensatz zur Papierversion, jede berechnigte Person Zugriff auf die erforderlichen Daten erhält. So können weitere behandelnde Ärzte oder Pfleger sich über den Behandlungsfortschritt eines Patienten erkundigen, ohne dass sie vor Ort sein müssen. Des Weiteren können simultan Statistiken oder Abrechnungsdaten erstellt werden. Ferner können auch Analysedaten direkt im Labor in die Datenbank eingepflegt werden. Befindet sich das Labor im Haus, erfolgt der Zugriff auf die elektronische Krankenakte direkt, ansonsten über Schnittstellen.

Phoenix besteht aus zwei Anwendungen, der Workstation und dem Designer. Die Workstation ist die Benutzeroberfläche im Produktivbetrieb, der Designer ist das Administrationswerkzeug, mit dessen Hilfe die Parametrierung (= Anpassung an lokale Gegebenheiten) durchgeführt wird.

Die Workstation bietet Bedienelemente für alle anfallenden Dateneingaben. Hier werden unter anderem die Behandlungs- oder Operationsdokumentationen, die Pflegeplanung, Laborbefunde oder die Medikation eingetragen.

2.5 Workflowmanagement

Grundlage des Workflowmanagement (WfM) sind Prozesse, in Form von Arbeitsabläufen. Diese Arbeitsabläufe werden in einer Ist-Analyse erhoben, können bei Bedarf optimiert und schließlich als Soll-Prozess modelliert werden. Die einzelnen Teilschritte eines Prozesses



Abbildung 2.5: Komponenten der elektronischen Krankenakte (Vgl. MCS-Phoenix [83]).

werden als Aktivitäten bezeichnet.

Im Rahmen des WfM werden die zuvor modellierten Prozesse von einem Informationssystem gesteuert. Die Aktivitäten selbst werden vom System als Black-Boxes betrachtet. Jede Aktivität kann über ein- und ausgehende Information in Form von digitalen Daten, realen Dokumenten oder Objekten verfügen. Bei digital vorliegenden Daten kann das WfMS die Weiterleitung zwischen Aktivitäten, bzw. von einem Bearbeiter zum nächsten vollständig übernehmen. Handelt es sich um reale Dokumente, im klinischen Umfeld z. B. vom Patienten unterzeichnete Einverständniserklärungen, kann das System die Bearbeiter über die Notwendigkeit der Dokumente informieren oder die Überprüfung veranlassen und die Abarbeitung, in Form von Checklisten sicherstellen. Gleiches gilt für die Durchführung von diagnostischen oder therapeutischen Maßnahmen.

Um einen Workflow in einem WfMS ausführen zu können, muss das Prozessmodell um

verschiedene Informationen angereichert werden. Zum einen sind dies Daten, die innerhalb der einzelnen Aktivitäten benötigt werden, zum anderen Daten, die den Ablauf steuern. Um diese Daten sauber unterscheiden zu können, wird oft vom Daten- und Kontrollfluss gesprochen. In diesem Projekt soll der Datenfluss nicht explizit modelliert werden, es wird von einer abstrakten Datenschicht ausgegangen, mit der die gesamte Interaktion stattfindet. Die Ablaufsteuerung findet implizit statt, d.h. die Komponenten erhalten eine Art Schaltlogik, die es ihnen ermöglicht bestimmte Parameter auszuwerten und entsprechend der Ergebnisse zu handeln.

Realisierung klinischer Pfade als Workflows

Für eine Umsetzung von Soll-Prozessen im klinischen Umfeld können klinische Pfade als Grundlage verwandt werden. Sie stellen gewissermaßen den Standardablauf für eine Behandlung dar, der als Pfad, bzw. Workflow, im System modelliert wird. Bei der Umsetzung soll es nicht mehr notwendig sein, bereits zur Modellierungszeit alle Eventualitäten mit einzubeziehen, vielmehr sollen dem Benutzer Möglichkeiten bereitgestellt werden, mit denen der modellierte Workflow zur Laufzeit an den vorliegenden Fall angepasst werden kann. Hierauf wird in Abschnitt 2.5 noch genauer eingegangen.

Besonderheiten im klinischen Umfeld

Im Gegensatz zu Produktionsbetrieben oder dem Dienstleistungssektor, wie Versicherungen, wo die einzelnen Aktivitäten an klar definierten Gegenständen (z. B. einer Schadensmeldung) orientiert sind, steht im Klinikalltag der Patient im Mittelpunkt. Hierbei stellt jeder Patient, für den Mediziner und das Pflegepersonal, eine völlig neue Situation dar. Somit muss das WfMS in jeder Situation in der Lage sein, Hilfestellungen in Form einer Management-Anwendung zu bieten. An dieser Stelle sei angemerkt, dass kein wissensbasiertes Expertensystem entwickelt werden soll, das den Behandelnden bei der Stellung einer Diagnose unterstützt. Das WfMS soll die Konsistenz des Informationsflusses und das Einbinden der jeweiligen Beteiligten übernehmen. So kann sichergestellt werden, dass Arbeitsabläufe zwar getreu eines Soll-Konzeptes durchgeführt werden, der Mediziner, als Verantwortlicher, jedoch jederzeit eingreifen kann. Ein wesentlicher Punkt der Modellierung von Prozessen im klinischen Umfeld ist die Tatsache, dass es unmöglich ist, sämtliche in Frage kommenden Variationen der Behandlung eines Patienten, im Vorfeld zu erfassen.

Notwendigkeit von Ad-hoc-Modifikationen

Viele Behandlungsalternativen oder vorhersehbare Abweichungen vom Standardfall sind zwar im Voraus ersichtlich, jedoch würde ein Modell, das all diese Optionen berücksichtigt, unüberschaubar groß werden. Folgende Beispiele zeigen Situationen auf, in denen Abweichungen vom Standardfall vorkommen können [37]:

- Ein Patient ist zu einer Operation angemeldet und vom behandelnden Arzt an den Anästhesisten übergeben worden. Der Operationsaal ist soweit vorbereitet. Der Anästhesist nimmt eine Routinebefragung vor. Hierbei erklärt der Patient, er habe bevor er sich zum Krankenhaus begeben hat, eine üppige Mahlzeit zu sich genommen. Dies führt dazu, dass die Operation gar nicht, bzw. nicht ohne weitere Maßnahmen durchgeführt werden kann, da in den sechs Stunden vor einer Operation weder etwas gegessen, noch etwas getrunken werden darf und nach Möglichkeit auch nicht geraucht werden sollte. Es bleiben folgende Alternativen:
 1. Es wird gewartet und die Operation wird um sechs Stunden verschoben. Dies ist natürlich nur möglich, wenn es sich um eine nicht-akute Operation, wie z. B. Gelenkoperationen handelt.
 2. Der Patient wird stationär aufgenommen oder wieder nach Hause entlassen und die Operation für den folgenden Tag neu eingeplant. Dieses Vorgehen ist ebenfalls nur bei nicht-akuten Operationen möglich.
 3. Angenommen es handelt sich um eine Operation einer akuten Erkrankung, wie z.B. einer Appendizitis, kann nicht gewartet werden und der Magen des Patienten muss entleert werden, damit die Operation wie geplant (bzw. mit geringer Verspätung) durchgeführt werden kann.
- Wesentlich häufiger ist der Fall, dass Patienten an einem fiebrigen Infekt erkranken, was eine Kontraindikation für eine Narkose darstellt. In diesem Fall muss ein Abklingen des Infekts abgewartet werden. Sofern es sich um einen stationär aufgenommenen Patienten handelt, wird der Klinikaufenthalt entsprechend verlängert, ansonsten wird der Patient neu einbestellt.
- Eine weitere mögliche Abweichung vom Standardfall wäre, dass der Patient bei der Routinebefragung des Anästhesisten angibt, dass Wunden immer ziemlich lange bluten. Hier muss der Patient zunächst auf eine Blutgerinnungsstörung hin untersucht werden, was dazu führt, dass die Operation, wie im obigen Beispiel, nicht wie geplant starten kann. Nun ergeben sich wieder mehrere Alternativen:
 1. Die Untersuchung auf die Blutgerinnungsstörung fällt negativ aus, alle Werte sind normal und die Operation kann doch wie geplant gestartet werden.
 2. Die Untersuchung auf die Blutgerinnungsstörung fällt positiv aus (der gleiche Fall ergibt sich, wenn der Patient die Einnahme von acetylsalicylsäurehaltigen Präparaten (z.B. Aspirin) angibt), nun muss wieder entschieden werden, wie weiter vorzugehen ist:
 - a) Der Patient wird nach Hause entlassen und der Operationstermin um fünf Tage nach hinten verschoben (wieder nur bei nicht-akuten Operationen möglich).
 - b) Dem Patienten werden Präparate zur Verstärkung der Blutgerinnung verabreicht, was nur bei Notfall-Operationen vorgenommen wird, da dieses Vorgehen ein höheres Risiko darstellt.

- c) Es werden mehr Blutkonserven als geplant vorgehalten, um mögliche Blutverluste kompensieren zu können, was aufgrund des Risikos ebenfalls nur bei Notfall-Operationen vorgenommen wird.

Allein an diesen, relativ einfachen Beispielen zeigt sich, dass ein Entscheidungsbaum, der die obigen beiden Beispiele behandelt und noch deren Kombinationen mit einbezieht, den Standardpfad explosionsartig aufblähen würde. Aus diesem Grunde muss ein Anwender in der Lage sein, den aktuellen Pfad mit vordefinierten Bausteinen ad hoc anpassen zu können. Gibt es jedoch für eine bestimmte Behandlung eine begrenzte Zahl von Standardpfaden, die unter bestimmten Bedingungen jedes Mal zur Anwendung kommen, so ist es möglich, diese als Varianten zu modellieren, die zu Beginn einer Behandlung ausgewählt werden. Beispielhaft sei hier eine abweichende Vorgehensweise für Patienten mit Diabetes genannt. Dieser Befund sollte relativ früh im Behandlungsverlauf feststehen und es kann somit z.B. eine Diabetiker-Variante erstellt werden, die entsprechend gewählt wird.

3 Lösungsansatz

Im folgenden Kapitel werden zunächst einige Designziele als Rahmenbedingungen für die Entwicklung einer Management-Anwendung erarbeitet. Im weiteren Verlauf werden einige etablierte Ansätze zur Modellierung und Optimierung von (Geschäfts-) Prozessen vorgestellt. Neben einer Vorstellung der Konzepte wird deren Tauglichkeit für das DynamicFlow-Konzept abgewogen. Im einzelnen wird auf folgende Konzepte eingegangen:

- Petri-Netze (Kapitel [3.2.1](#))
- Ereignisgesteuerte Prozessketten (EPK) (Kapitel [3.2.1](#))
- ADEPT (Kapitel [3.2.3](#))
- AgentWork (Kapitel [3.2.4](#))
- Controlflow-Constrains (Kapitel [3.2.5](#))
- Dataflow-Constraints (Kapitel [3.2.6](#))
- Workflow-Patterns (Kapitel [3.2.7](#))

Die Argumentation schließt mit der Einführung eines eigenen DynamicFlow-Konzepts.

3.1 Designziele

Um das System so zu gestalten, dass für Anwender aus informatik fernen Disziplinen möglichst wenig Einarbeitungsaufwand abverlangt wird, wurden verschiedene Kriterien aufgestellt, die in den nächsten Abschnitten beschrieben werden.

3.1.1 Identifizierung und Modellierung von Prozessen

Der avisierte Einsatzbereich im Krankenhaus führt dazu, dass die Benutzbarkeit des Systems ohne großen Einarbeitungsaufwand gegeben sein muss. An der Erstellung eines Workflows sind z. B. eine Reihe von Akteuren beteiligt. Mediziner prüfen die medizinische Richtigkeit der Vorgehensweise, Sachbearbeiter der Verwaltung müssen die rechtlichen sowie organisatorischen Randbedingungen abklären. Techniker müssen dafür sorgen, die fachlich spezifizierten Workflows technisch umzusetzen.

Als Arbeitsgrundlage wurden reale Prozesse im Krankenhaus analysiert und modelliert. Die Modellierung erfolgte in Form von ereignisgesteuerten Prozeßketten (EPK). Die PG DynamicFlow hat zum Ziel, einen praxisgerechten Lösungsansatz für die in der Einleitung besprochenen Anforderungen zu finden.

Konkret wird hierbei auf folgenden Ebenen gearbeitet:

- Identifikation und Beschreibung eines Metamodells für adaptive Workflows
- Umsetzung von bestehenden (EPK basierenden) Workflows in das neue Modell
- Implementierung des Metamodells und Einbettung dieser Implementierung in einen graphischen Designer und eine Workflow-Engine (Die Anwendung)

Die vorgestellten Anforderungen sind die Grundlagen, auf denen Design und Implementierung sowohl des Metamodells, als auch der Anwendung, gründen.

3.1.2 Funktionale Ziele

Funktionale Anforderungen beschreiben, *was* ein Programm, Modell oder Gerät leisten oder *ausführen* soll (vgl. [90, S. 9-10]).

Die hauptsächlichsten funktionalen Anforderungen lassen sich in folgende Punkte zusammenfassen:

Grundlegende Anforderungen an das Workflow-Model

Es werden die folgenden, grundlegenden Eigenschaften des Workflow-Modells gefordert

- Das Workflow-Model muss mächtig genug sein, um klinische Pfade abzubilden.
- Das Workflow-Model muss ausführbar sein.
- Das Workflow-Model muss Ad-hoc-Modifikation unterstützen.
- Die strukturelle Korrektheit eines Workflows muss überprüfbar sein, d. H. je nach Anforderung an die Korrektheit (s. 3.3.10) beispielsweise: Er muss abgeschlossen werden können und alle Teilpfade müssen potentiell durchlaufen werden können.

Diese ersten drei Forderungen ergeben sich aus den in Kapitel 1 vorgestellten Anforderungen. Die Forderung, die strukturelle Korrektheit eines Workflows prüfen zu können ermöglicht es, die von den Benutzern erstellten Workflows grundlegend zu überprüfen.

Es wird nicht gefordert, dass der Workflow auf Deadlocks oder nicht erreichbare Aktionen geprüft werden kann. Eventuell auftretende Deadlocks können durch den Anwender ad hoc gelöst werden. Die Freiheit in der Modellierung wurde hier wichtiger erachtet, als (zwingend) überprüfbare Eigenschaften der Laufzeit.

Unterstützung der Modellierung von Datenflüssen

Weiterhin werden bestimmte Eigenschaften an die Unterstützung von Datenflüssen gestellt.

- Ein Datenfluss muss modellierbar sein.
- Der Datenfluss darf aber nicht erzwungen werden, d.h. eine Modellierung ohne stringent modellierten Datenfluss muss möglich sein.
- Die *Produktion* und die *Konsumierung* von Daten muss ausgezeichnet werden können.

Auch hier wurde eine Abwägung zwischen dem Nutzen und den Nachteilen, die durch einen expliziten und erzwungenen Datenfluss entstehen, vorgenommen. Unsere Analyse der bestehenden Workflows und des Klinikalltags hat gezeigt, dass die im Klinikalltag anfallenden Daten selten in einem Format vorliegen, das sich zur automatischen Auswertung durch die Workflow-Engine eignet. Auf der anderen Seite erhöht die explizite Modellierung des Datenflusses die Anforderungen an den Modellierer. Als Kompromiss wird ein impliziter Datenfluss gefordert. Das heisst, die Produktion und der Bedarf von Daten kann in einem Workflow angezeigt werden. Es wird aber nicht erzwungen, dass die Produktion des Datums A vor der Konsumierung des Datums A stattfindet. Das Workflow-Model muss sogar unterstützen, dass Datum A (z.B. ein erhobener Blutwert) konsumiert wird, obwohl kein Produzent ausgezeichnet ist. Dies kann auftreten, wenn Daten verwendet werden deren Erhebung nicht Teil der Workflowmodellierung ist, beispielsweise regelmäßig durchgeführte Pulskontrolle. Konflikte und Deadlocks, die aus diesen Situationen entstehen können, müssen durch menschliche Intervention, gelöst werden.

Visualisierung

Die hauptsächlichen Nutzer des Workflow-Models werden keine Experten für Workflowmodellierung sein. Deshalb ist es wichtig, dass das Workflow-Model noch folgenden Anforderungen genügt:

- Das Workflow-Model muss es ermöglichen, verständliche Workflows zu modellieren.
- Das Workflow-Model muss verschiedene *Sichten* für z. B. Designer, Entwickler oder Fachpersonal unterstützen.
- Das Workflow-Model muss sich für die Visualisierung des Workflows für unterschiedliche Anwendergruppen eignen.

Weitere Anforderungen

Weiterhin müssen die folgenden Anforderungen bei Verwendung des Workflow-Modells umsetzbar sein:

- Ein Rollen- und Rechtekonzept
- Nachvollziehbarkeit: Unterstützung für eine revisionssichere Protokollierung, sowohl für strukturelle Änderungen, als auch für die Ausführung der Aktivitäten
- Das Workflow-Modell sollte bei Bedarf durch weitere Konstrukte erweiterbar sein.

3.1.3 Begründung

Die folgenden Abschnitte betrachten das bearbeitete Problemfeld und zeigen identifizierte Use-Cases.

Konträr zu Workflows in streng kontrollierten Umgebungen, wie z. B. bei Produktionsprozessen, sind Abweichungen vom Standard-Workflow in Krankenhäusern wesentlich häufiger. Abweichungen können aus diversen Gründen vorkommen (vgl. Kapitel 2.2). Tritt eine solche Abweichung auf, dann muss der Workflow angepasst werden. Konkret muss die *Instanz* des Workflows angepasst werden, bei der die Abweichung aufgetreten ist.

Geht man mehr ins Detail, so kann man noch zwischen *geplanten Abweichungen*, sogenannten *Varianten* und *ungeplanten Abweichungen* unterscheiden. Der Aspekt *Varianten* wird in Abschnitt 3.3.9 eingehender besprochen.

Die Themen „*Verschiedene Sichten*“ und „*Ad-hoc-Modifikationen*“ sind zentrale Themen des ersten Semesters gewesen und werden in den nächsten Abschnitten beschrieben.

Verschiedene Sichten

In Abschnitt 3.1.2 wird aufgezeigt, dass an der Entwicklung und Ausführung eines Workflows eine Reihe von Gruppen beteiligt sind. Diese Gruppen haben nicht nur unterschiedliche *Ziele* und *Interessen*, sie haben in der Regel auch einen voneinander verschiedenen *Wissenshintergrund*.

Um allen Gruppen gerecht zu werden, werden verschiedene *Sichten* auf den Workflow unterstützt. Die drei als wichtigste identifizierte Gruppen sind *technische Entwickler*, *fachliche Entwickler (Modellierer)* und *Endanwender (Klinikpersonal)*. Entwickler kümmern sich um die technische Umsetzung eines Workflows, Modellierer definieren die fachlichen Grundlagen und definieren die Workflows. Das Klinikpersonal (Ärzte, Pflege- und Verwaltungspersonal) führen die sich aus dem Workflow ergebenden Aufgaben aus.

Folglich werden auch mindestens drei verschiedene Sichten unterstützt

- Technische Sicht (Konstruktion, *technische Entwickler*)
- Fachliche Sicht (Konfiguration, *fachlicher Entwickler*)
- Aufgabensicht (Ausführung, Bedienung, *Endanwender*)

Das Hauptaugenmerk der PG liegt auf den letzten beiden Punkten: der fachlichen- und der Benutzer-Sicht.

Die *Technische Sicht* erlaubt es den Entwicklern technische Elemente (Z. B. BPEL-Code) so anzulegen und darzustellen, dass die fachlichen Anwender diese Fragmente in der *Fachlichen Sicht* nutzen können, um ihre Workflow-Modellierung durchzuführen. Irgendwann wird der Workflow ausgeführt. Die dabei definierten Tätigkeiten (Blutdruck messen, Patient lagern, ...) sind in der Regel so genannte *Human Tasks*, also Aufgaben, die von Menschen durchgeführt werden. Diese Nutzer des Systems werden *Endanwender* genannt. Sie nutzen die *Aufgabensicht* des Systems. Hier können sie Aufgaben durchführen und die Ergebnisse dieser Durchführung protokollieren.

Die im Rahmen eines Workflows durchgeführten Aktivitäten haben in der Regel nicht nur Auswirkungen auf die reale Welt (z. B. der Patient wird gelagert), viele dieser Aktivitäten *konsumieren* oder *produzieren Daten*. Als ein Beispiel mag dienen, dass die Aktivität „*Blutdruck messen*“ offensichtlich den Wert des Blutdruckes eines Patienten bestimmt. Diese Aktivität *produziert* also ein Datum. Die Aktivität „*Arztbrief erstellen*“ produziert nicht nur ein Datum (den Arztbrief), sie *konsumiert* auch Daten, wie den erhobenen Blutdruck.

Ad-hoc-Modifikationen

Die Unterstützung von Ad-hoc-Modifikationen an laufenden Workflows ist das hauptsächliche Ziel der PG DynamicFlow. Konventionelle Workflow-Systeme unterstützen die Änderung von laufenden Workflow-Instanzen nicht, oder nur sehr eingeschränkt. Im Regelfall ist der Workflow die treibende Kraft hinter einem Prozess wie *Herstellung eines Autos*. Jedes Auto in der entsprechenden Fabrik wird nach demselben Workflow fertiggestellt. Diese Workflows können zwar Unterschiede untereinander aufweisen, aber die einzelnen *Instanzen* eines Workflows sind in der Regel immer strukturell gleich. Soll, um beim Beispiel Auto-Produktion zu bleiben, Wagen A zusätzlich mit einem Partikelfilter ausgestattet werden, so wird der Umstand, dass ein Partikelfilter eingebaut wird oder eingebaut werden kann, schon bei der Erstellung der Workflow-Struktur bedacht. Vor der Produktion des ersten Wagens wird der komplette Workflow mit allen möglichen alternativen Pfaden festgelegt, eine nachträgliche Änderung ist (ohne Beweis) in der Regel mit signifikanten Kosten verbunden.

Im medizinischen Umfeld ist eine solche Vorgehensweise nicht vorstellbar. Der Zustand des Patienten ist der dominierende Faktor. Erlebt ein Patient einen anaphylaktischen Schock aufgrund einer Medikamentengabe, ist ein Festhalten am bestehenden Workflow nicht nur nicht sinnvoll, es würde sogar das Leben des Patienten gefährden. Der Workflow muss in der laufenden Instanz geändert werden. Eine solche Änderung wird als Ad-hoc-Modifikation

bezeichnet (Abschnitt 2.5 beschreibt die Gründe detailliert). Eine genauere Betrachtung des Themas ergibt, dass zwei verschiedene Arten der Änderung betrachtet werden müssen:

- Geplante Änderungen am Workflow (*Varianten*)
- Ungeplante Änderungen am Workflow (*ad hoc*)

Um das Workflow-Model möglichst einfach zu halten, werden sowohl Varianten, als auch die beschriebenen ad hoc Änderungen über dieselben Strukturen und Mechanismen abgebildet. Kurz zusammengefasst, sind Varianten vorgefertigte ad hoc Änderungen, die en bloc in eine Workflow-Instanz eingearbeitet werden.

3.1.4 Ergonomie-Anforderungen

Wird ein System nicht verwendet, bringt es auch keinen Nutzen. So offensichtlich dieser Satz ist, so oft trifft man auf Anwendungen, die von den designierten Benutzern abgelehnt werden. Es ist also wichtig, dass ein (neu) entwickeltes oder eingeführtes System vom Benutzer akzeptiert und verwendet wird. Ziel muss es sein, die Verwendbarkeit und Akzeptanz des Systems und damit auch Seine Nutzung zu verbessern.

In seinem Buch *Flow: The Psychology of Optimal Experience* [35] beschreibt Mihaly Csikszentmihaly einen Zustand namens *Flow*. In diesem Zustand kann ein Mensch gut, gerne und effizient arbeiten. Dieser Zustand wird vor allem dadurch hervorgerufen, dass der Mensch Aufgaben hat, deren Lösung fordernd sind, aber nicht zu fordernd. Sind die an den Menschen gestellten Aufgaben zu fordernd, dann entsteht Frust. Der Auslöser für den Frust wird als störend und hinderlich empfunden, der Mensch versucht den Auslöser zu meiden.

Transferiert auf eine Software bedeuten diese Erkenntnisse: Der Benutzer muss mit der Software intellektuell arbeiten können, ohne aber von ihr (zu sehr) behindert zu werden. Die Erfahrung bestätigt diese Vermutung. Kann der Benutzer mit seiner Software auf einem Niveau kommunizieren, dass eben in diesem Kanal zwischen *zu leicht* und *zu schwer* liegt, dann „geht die Arbeit leichter und schneller von der Hand“. Wird der Benutzer hingegen von seiner Software gefrustet, dann wird er diese nicht verwenden und nach Möglichkeiten suchen, die Software zu meiden. Dies darf nicht vorkommen. Vor allem nicht, wenn man bedenkt, dass die Software nicht als (reine) Arbeitserleichterung eingesetzt wird, sondern für das Krankenhaus lebenswichtig sein kann.

Zeit und Aufmerksamkeit sind im ohnehin schon komplexen Klinikalltag eine knappe Resource. Sollen die Workflow-Systeme also verwendet werden, ist dafür zu sorgen, dass jede der in 3.1.3 aufgeführten Benutzergruppen optimal mit dem System interagieren kann.

Die kritischsten Interaktionspunkte mit dem System sind

- Das Durchführen von Änderungen durch den fachlichen Benutzer (Arzt)

- Durch den Workflow getriebene Ausführung von Aktivitäten durch die Benutzer (Schwestern).

Fachliche Benutzer werden vor allem durch die Unterstützung von Sichten und der Vorkonfektionierung von Varianten oder Aktionsplänen (3.3.9) unterstützt. Besonders kritisch ist die Unterstützung der Benutzer, in der Regel also der Krankenschwestern. Die Teams der einzelnen Stationen sind oft gut aufeinander eingespielt und arbeiten sehr effizient. Diese Effizienz lässt sich aber nur erhalten, wenn die Schwestern in gewissem Rahmen eine Entscheidungskompetenz über die Reihenfolge der durchgeführten Aktivitäten behalten, wobei Ärzte oder anderes administratives Personal ebenfalls als fachliche Benutzer zu sehen sind. Befinden sich beispielsweise zufällig zwei Schwestern in einem Zimmer, ist es oft angebracht den Patienten Meyer jetzt schon neu zu lagern, obwohl er nach Plan erst in einer halben Stunde gelagert werden soll. Besagt der Workflow, dass einem Patienten der Puls gemessen werden soll und danach der Blutdruck, dann wäre es im höchsten Masse ineffizient, würde die Schwester zwischen diesen Schritten zum PC gehen.

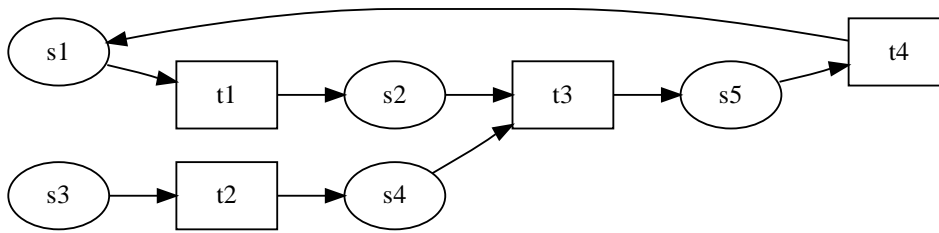
3.2 Nutzbarkeit existierender Ansätze

Die Festlegung, welche Ansätze als Basis für das von der Projektgruppe konzipierte Workflow-Model verwendet werden, erfordert es, bestehende Ansätze zu evaluieren und zu vergleichen. Im folgenden Kapitel werden einige der untersuchten Ansätze vorgestellt. Dabei handelt es sich teilweise um sehr grundlegende, mathematische Konstrukte wie Petrinetze oder Prozessalgebren, teilweise auch um bestehende Forschungsarbeiten aus dem Bereich klinische Workflows.

3.2.1 Petri-Netze

Petrinetze [17] sind mathematische Modelle, mit deren Hilfe sich nebenläufige Systeme beschreiben lassen. Sie wurden 1962 von Carl Adam Petri eingeführt[84]. Petrinetze sind bipartite, gerichtete Graphen, die mit Schaltregeln versehen sind. Sie sind selber keine Beschreibungssprache für Workflows, werden jedoch als Basis vieler WfMs verwendet (siehe z.B. [97]).

Die beiden Knotentypen sind *Stellen* und *Transitionen*. Eine Stelle, traditionell als Kreis dargestellt, kann *Marken* enthalten. Transitionen, durch Rechtecke dargestellt, sind die aktiven Elemente von Petrinetzen. Sind die Vorbedingungen einer Transition gegeben, dann kann sie *schalten*. Dabei werden aus dem *Vorbereich* der Transition Marken entnommen und im *Nachbereich* Marken erzeugt.



Formale Definition

Ein Petri-Netz ist ein 6-Tupel (S, T, F, K, W, m_0) , wobei das Netz (S, T, F) den bipartiten und gerichteten Graphen beschreibt.

1. S , nichtleere Menge von Stellen $S = s_1, s_2, \dots, s_{|S|}$
2. T , nichtleere Menge von Transitionen (Übergängen) $T = t_1, t_2, \dots, t_{|T|}$
3. F , nichtleere Menge der Kanten (Flussrelation) $F \subseteq (S \times T) \cup (T \times S)$
4. K , Kapazitäten der Plätze, Kapazitätsfunktion $K : S \rightarrow \mathbb{N} \cup \infty$
5. W , Kosten der Kanten, Gewichtsfunktion $W : F \rightarrow \mathbb{N}$
6. m_0 Startmarkierung $m_0 : S \rightarrow \mathbb{N}$

Die Mengen der Stellen S und Transitionen T sind disjunkt.

Den aktuellen Zustand eines Petrinetzes nennt man die (aktuelle) *Markierung*. Sie beschreibt für jede Stelle, wieviele Marken auf ihr liegen. m_0 ist die Start- oder Anfangsmarkierung des Petrinetzes. Die Anzahl der Marken auf einer Stelle $s \in S$ wird durch $m(s)$ beschrieben. Die Stelle s darf zwischen 0 und $k(s)$ Marken enthalten, man sagt auch, sie hat eine *Kapazität* von $k(s)$.

Für eine Transition $t \in T$ sind folgende Mengen (von Stellen) von besonderem Interesse:

1. Der Vorbereich $\bullet t = \{s \in S \mid (s, t) \in F\}$, also alle Stellen, von denen eine Kante zur Transition t führt und
2. Der Nachbereich $t \bullet = \{s \in S \mid (t, s) \in F\}$, also alle Stellen, zu denen eine Kante von der Transition t aus führt.

Schalten von Transitionen

Eine Transition t kann *schalten*, wenn sie *aktiviert* ist. Dies ist der Fall, wenn

1. $\forall s \in \bullet t : m(s) \geq W(s, t)$ – Es liegen genügend Marken in jeder Stelle des Vorbereichs

2. $\forall s \in (t \bullet \setminus \bullet t) : K(s) \geq m(s) + W(t, s)$ – In jeder Stelle des Nachbereichs ist genügend Platz für die neu erzeugten Marken. Hier werden nur Stellen betrachtet, die nicht im Vorbereich der Transition liegen
3. $\forall s \in t \bullet \cap \bullet t : K(s) \geq m(s) + W(t, s) - W(s, t)$ – dito, nur für Stellen, die im Vor- und Nachbereich der Transition liegen

Zu beachten ist, dass die Kanten *gewichtet* sind. Diese Gewichtung bestimmt, wieviele Marken entnommen bzw. erzeugt werden.

Eine aktivierte Transition *muss* nicht schalten. Dadurch zeigen Petrinetze ein nicht-deterministisches Verhalten.

Beim Schaltvorgang selber werden Marken aus dem Vorbereich entnommen und Marken im Nachbereich erzeugt. Formal bedeutet das

$$m'(s) = \begin{cases} m(s) - W(s, t) & \text{falls } s \in \bullet t \wedge s \notin t \bullet \\ m(s) + W(t, s) & \text{falls } s \notin \bullet t \wedge s \in t \bullet \\ m(s) - W(s, t) + W(t, s) & \text{falls } s \in \bullet t \wedge s \in t \bullet \\ m(s) & \text{sonst} \end{cases}$$

Dargestellt wird der Schaltvorgang durch $m[t > m']$, gelesen „Die Markierung m wird durch Schalten von t in m' überführt“. m' ist die *Folgemarkierung* von m , $M = \{m' | m[t >^* m']\}$ ist die Menge aller, durch beliebig häufiges Schalten beliebiger Transitionen von m erreichbaren Markierungen, wobei $m[t >^*$ das mehrfache Schalten von t beschreibt.

Die Marken eines Petri-Netzes sind in ihrer einfachsten Form voneinander nicht unterscheidbar. Für komplexere, aussagekräftigere Petri-Netze sind *Markeneinfärbungen*, *Aktivierungszeiten* und Hierarchien definiert worden.

Aussagen über Petrinetze

Über Petrinetze können diverse Aussagen getroffen werden. Eine wichtige Eigenschaft von Petrinetzen ist die *Lebendigkeit*. Ein lebendiges Netz wird z. B. nie in einen Zustand kommen, an dem keine weitere Schaltung möglich ist.

Eine *Transition* heißt

tot, falls sie unter keiner Folgemarkierung aktiviert ist.

aktivierbar, falls sie unter mindestens einer Folgemarkierung aktiviert ist.

lebendig, falls sie in jeder erreichbaren Markierung aktivierbar ist.

Ein *Petrinetz* heißt

tot, falls alle Transitionen tot sind.

todesgefährdet, falls das Petri-Netz unter einer Folgemarkierung tot ist.

verklemmungsfrei oder schwach lebendig, falls es unter keiner Folgemarkierung tot ist.

(stark) lebendig, falls alle Transitionen lebendig sind.

Erreichbarkeit: Eine Markierung m' heißt von m aus *erreichbar*, falls $m' \in m[>^*$.

Beschränktheit: Ein Petri-Netz heißt n -beschränkt, wenn es eine Schranke n gibt, so dass nie mehr als n Marken in einer Stelle liegen.

Sicherheit: Ein Petri-Netz heißt sicher, falls es 1-beschränkt ist.

Konflikt: Es besteht ein Konflikt bei einer nicht nebenläufigen Aktivierung von zwei Transitionen. Im Vorbereitungsbereich bedeutet das, dass zwei Transitionen die gleiche Marke benötigen um zu schalten. Im Nachbereich, wo man den Konflikt auch als *Kontakt* bezeichnet, sind es zwei Transitionen, die Marken erzeugen können, aber die Kapazität nicht für beide ausreicht.

Erweiterungen von Petrinetzen

In ihrer ursprünglichen Form sind Petrinetze nicht ausdrucksstark genug, um eine Reihe von Problemen zu beschreiben. So sind Petrinetze z. B. völlig zeitlos. Temporale Aspekte können mit normalen Petrinetzen nicht abgebildet werden. Petrinetze können auch keine Informationen *von außen* aufnehmen, eine Reaktion auf ein externes Ereignis ist also auch nicht möglich. Aus diesen und noch anderen Gründen wurden viele Erweiterungen für Petrinetze vorgeschlagen.

Einige dieser Erweiterungen sind z. B.:

Prioritäten werden als Zahlen, beginnend mit 1, neben einer Transition notiert. Wenn zwei zeitlose Transitionen aktiviert sind, schaltet die mit der höheren Priorität.

Hemmende Kanten verbinden Stellen mit Transitionen. Wenn eine hemmende Kante durch eine Marke in ihrer Ausgangsstelle aktiviert ist, kann die verbundene Transition nicht feuern, auch wenn alle ihre anderen eingehenden Kanten aktiviert sind.

Farbige Petrinetze Farbige Petri-Netze erweitern Petrinetze um Marken mit verschiedenen Farben. Während Marken bei normalen Petri-Netzen nicht unterschieden werden können, ist dies durch die Färbung der Marken möglich.

Nutzbarkeit für die PG DynamicFlow: Untersuchte Aspekte

Dieser Abschnitt untersucht die Nutzbarkeit von Petrinetzen für die Projektgruppe. Betrachtet werden die folgenden Aspekte

1. Wie wird Flexibilität im Kontrollfluss unterstützt?
2. Wie wird Flexibilität im Datenfluss unterstützt?
3. Werden Ad-hoc-Modifikationen unterstützt, wenn ja: Wie?
4. Werden verschiedene Sichten unterstützt?
5. Ist die Modellierung durch Fachanwender vorgesehen?
6. Ist eine besondere Unterstützung für Semantik vorgesehen?

Die Auswahl der betrachteten Themen ergibt sich aus den gestellten Anforderungen und erlaubt so eine Einschätzung, in wie weit Petrinetze als Basis für das WfM dienen können.

Flexibler Kontrollfluss

Die Ausführung eines Petri-Netzes ist *nicht deterministisch*, da aktivierte Transitionen nicht feuern müssen. Diese Eigenschaft macht Petrinetze gut geeignet um parallele Vorgänge abzubilden. Eine weitergehende Flexibilisierung im Kontrollfluss ist bei Petrinetzen nicht vorgesehen. Insbesondere sind *externe Ereignisse* nicht vorgesehen.

Unterstützung von Ad-hoc-Modifikationen

Petri-Netze sind vergleichsweise grundlegende mathematische Modelle. In ihren ursprünglichen Formen (B/E bzw. Stellen-/Transitionen Netze) haben Petrinetze eine fixe Struktur (F, S, T) . Prinzipiell spricht nichts gegen die Ad-hoc-Modifikationen von Petrinetzen. Van der Aalst et al. z. B. nutzen Petrinetze als Basis für ihre Ad-hoc-Workflows [97].

Verschiedene Sichten

Hierarchische Petri-Netze erlauben es, ein Petrinetz in verschiedenen Verfeinerungen/Vergrößerungen zu betrachten. Bei einer Vergrößerung werden Teilnetze durch einzelne Stellen oder Transitionen ersetzt. Ein Teilnetz, dessen *Oberfläche* nur aus Stellen besteht, kann in einer Vergrößerung durch eine Stelle ersetzt werden. Alle Kanten, die vorher eine Stelle s_i der Oberfläche mit einem Knoten (s'_i) ausserhalb des zu ersetzenden Teilnetzes verbunden haben, verbinden nun den neuen Knoten s_n mit dem Knoten s'_i . Selbiges gilt auch für Teilnetze mit einem Rand aus Transitionen. Vordefinierte Vergrößerungen können eine für den Fachanwender vereinfachte Sicht auf den Workflow bieten.

Benatallah et al. verwenden Hierarchische Petrinetze für ihren Workflow-Modeller *HiWorD*. Neben Hierarchischen Petrinetzen unterstützt HiWorD auch eine Form der Ausnahmebehandlung durch „Recovery transitions“ [39].

Datenfluss

Der Datenfluss bei Petri-Netzen wird durch die Marken und das Netz selbst vorgegeben. Einen dedizierten, vom Kontrollfluss unabhängigen Datenfluss gibt es bei Petrinetzen nicht. Es gibt verschiedene Erweiterungen (Gefärbte Petrinetze oder z. B. [46]) von Petrinetzen, die sich mit der Problematik des Datenflusses beschäftigen. Es ist jedoch durchaus möglich, ein Petrinetz als Datenflussdiagramm zu betrachten. Dabei werden bestimmte Stellen als Daten-Entität gekennzeichnet. Transitionen im Nachbereich der Stellen *konsumieren* das Datum, Transitionen im Vorbereich der Stelle *produzieren* das Datum.

Modellierung durch Fachanwender

Petrinetze sind mathematische Modelle die nicht für Fachanwender durchschaubar sind. Es gibt auch keine explizite Sicht für Fachanwender.

Semantiken

Durch die Erweiterung von Petrinetzen durch LTS (*Labeled Transition Systems*) ist es möglich Knoten und Kanten mit *Labeln* zu versehen. Diese *Label* können für eine semantische Interpretation des Netzes genutzt werden.

Nutzbarkeit für die PG DynamicFlow: Ergebnis

Petrinetze sind eine interessante Basis für das WfM. Aus diesem Grund fließen auch einige der Konzepte in das WfM ein. Das WfM ist ebenfalls als Netz ausgelegt und unterstützt die parallele Ausführung von Aktionen.

Die mangelhafte Unterstützung für ad hoc Änderungen, die Komplexität von farbigen Petrinetzen und die nicht explizit vorhandene Unterstützung für verschiedene Sichten schließen Petrinetze jedoch als alleinigen Lösungsansatz aus.

3.2.2 Ereignisgesteuerte Prozessketten

In den 70er Jahren kamen in Produktions- und Dienstleistungsbetrieben Forderungen nach einem effizienteren Einsatz von Ressourcen durch qualitativ höherwertig modellierte Prozesse auf. Um diese komplexen Geschäftsprozesse bzw. Geschäftsprozess-Ketten zu modellieren, entwickelte Professor August-Wilhelm Scheer (Gründer des Instituts für Wirtschaftsinformatik in Saarbrücken) in den 90er Jahren die ereignisgesteuerten Prozessketten auf Basis von Petrinetzen. Zu der theoretischen Arbeit über die EPK entwickelte Scheer das ARIS-Toolset zur Modellierung. Daraus entstand das von Scheer gegründete Spin-off Unternehmen, die IDS-Scheer AG (IDS = Integrierte Datenverarbeitungs Systeme)

Ziel war es eine Modellierungsgrundlage zu schaffen, die Rollen, Ressourcen, Daten und Schnittstellen einzubinden und zu definieren. EPK dienen der Unterstützung bei der Dokumentation und Optimierung vorhandener Prozessabläufe sowie Simulation und Kommunikation bei der Geschäftsprozessmodellierung in der Industrie.

Modellierungselemente der EPK

Eine EPK besteht aus einer streng iterierenden Folge aus Ereignissen und Funktionen. Jede EPK beginnt mit einem Ereignis und endet mit einem Ereignis. Ereignisse sind passive Elemente und stellen einen zeitpunktbezogenen Sachverhalt (Zustand) ohne Entscheidungskompetenz dar. Funktionen sind zeitbenötigende, aktive Elemente mit Entscheidungskompetenz.

Verwendung für DynamicFlow: Unterstützung von Ad-hoc-Modifikationen

Da EPK zur Erfassung, Modellierung und Optimierung von Geschäftsprozessen dienen, bieten sie durch die freie Kombination der einzelnen Elemente ein ausreichendes Maß an Flexibilität zur Modellierungszeit. Sie dienen jedoch nicht dazu, um ausführbare Modelle zu erzeugen. Dies führt dazu, dass eine nachträgliche Ad-hoc-Modifikation nur auf rein syntaktischer Ebene verifiziert werden kann.

Datenfluss

Der Datenfluss innerhalb einer EPK wird nur über Annotationen der am Prozess beteiligten Dokumente dargestellt. Somit existiert kein direkter Datenfluss innerhalb der Funktionen oder über die Grenzen von Funktionen hinweg.

Sichten

Eine direkte Unterstützung verschiedener Sichten ist nicht vorgesehen, jedoch besteht die Möglichkeit, Prozessschritte zu kapseln oder zu hierarchisieren, indem einer Funktion ein weiterer Sub-Prozess zugeordnet wird, der nur auf Anforderung eingeblendet wird. Weiter ist nicht davon auszugehen, dass Fachanwender informatikferner Disziplinen auf Anhieb in der Lage sind, Prozessabläufe mit Hilfe von EPK zu modellieren.

Vor- und Nachteile

Vorteile:

- Klare Struktur

- Flexibel in der Modellierung
- Einbinden von Ressourcen (Daten, Rollen, Dokumente)
- Bedingungen gut erfassbar (Ereignisse abfragen)
- Darstellung von Gesamtabläufen

Nachteile:

- kaum zeitliche Aspekte
- schlechte Anpassungsmöglichkeit
- nicht gut ad-hoc veränderlich

3.2.3 ADEPT





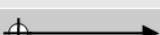


Kantentyp	Graphische Darstellung
CONTROL_E	
CONTROL_E (Default-Zweig)	
SOFT_SYNC_E	
STRICT_SYNC_E	
LOOP_E	
FAILURE_E	
PRIORITY_E	

Abbildung 3.1: Kantentypen in ADEPT [92].

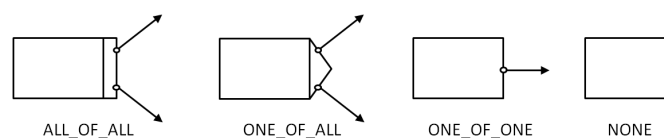


Abbildung 3.2: Knotenausgänge in ADEPT [92].

ADEPT ist ein WfMS, das auf Ad-hoc-Modifikationen ausgelegt ist. Ein Workflow in ADEPT besteht aus Knoten und Kanten. Es gibt verschiedene Kanten- (siehe Abbildung 3.1) und Knotentypen (siehe Abbildung 3.2). Zudem gibt es Split- und Join-Knoten, sowie Kanten, die den Kontrollfluss definieren, Synchronisationen vorgeben oder Schleifen

beschreiben. Diese Unterscheidung in verschiedene Kantentypen ermöglicht es, Schleifenkonstrukte zu verwalten. Ausgehend von einem Start und einem Endknoten werden in ADEPT neue Knoten hinzugefügt. Einfügeoperationen beziehen sich immer auf einen definierten Block, in den sie eingefügt oder zu dem sie parallel geschaltet werden. Somit ist die Struktur in ADEPT-Workflows immer in Blöcken organisiert. Durch die Konstruktionsweise und die Aktivierungssemantik der Kanten wird in ADEPT sichergestellt, dass sich ergebende Workflows immer syntaktisch korrekt sind. Zudem lassen sich in ADEPT Datenflüsse explizit definieren [92]. Die Grundmaxime von ADEPT ist, dass es keine Fehler zur Laufzeit geben darf, dies soll schon durch Einschränkungen während der Modellierung, unter anderem durch die Bildung von Blockstrukturen, verhindert werden.

ADEPT hat aufgrund seiner Konstruktionsweise einfache Möglichkeiten von außen in den Workflow einzugreifen. Der Benutzer wird beim Einfügen neuer Knoten unterstützt. Für kleinere Änderungen, die nicht unbedingt den Aufbau eines neuen Workflows beschreiben, ist es so möglich leicht durch eine natürliche Formulierung die Position des neuen Knotens anzugeben, wobei ADEPT für die syntaktisch korrekte Positionierung des Elementes sorgt. Der explizite Datenfluss und die syntaktische Korrektheit, sowie das ausgefeilte System, mit dem Änderungen an Workflowdefinitionen sich auch auf Instanzen (solange diese bestimmten Bedingungen genügen) auswirken, sind ein großer Vorteil von ADEPT. Gleichzeitig kann der explizite Datenfluss auch als Nachteil betrachtet werden, da es so nicht (oder nur unter Pseudo-Modellierung) möglich ist Modellierungslücken zu erlauben. Hier ist es z. B. nicht möglich, auf die Modellierung der Herkunft von Daten zu verzichten, sofern diese im weiteren Verlauf des Workflows verwendet werden sollen. Des Weiteren kann die automatische Positionierung der Elemente eine leicht verständliche Modellierung behindern.

3.2.4 AgentWork

AgentWork implementiert Ad-hoc-Modifikationen an Workflows durch die Anwendung von Event-Condition-Action-Regeln (ECA). Dabei haben die ECA-Regeln folgenden Aufbau: Wenn ein Ereignis (Event) eintritt wird geprüft, ob hierdurch bestimmte Bedingungen erfüllt werden. Treffen diese zu, werden bestimmte Modifikationen am Workflow vorgenommen. Die Definition dieser Regeln bedient sich dabei einer Sprache die Bedingungen inklusive temporalen Bedingungen auswerten kann [85]. Abbildung 3.3 zeigt wie sich die ECA-Regeln in das Workflowmodell integrieren. Agentworks reagiert also auf „vormodellierte“ Ereignisse und passt den Workflow an diese an. Schwierigkeiten können sich ergeben, wenn mehrere Modifikationen an einem Workflow notwendig werden, deren Auftreten zu Überschneidungen führen. Zudem ist dieser Ansatz problematisch, weil die Menge der möglichen Abweichungen schon durch die Menge der vormodellierten ECA-Regeln eingeschränkt ist. Agentworks ist also eher als ein WfMS mit automatisierten Varianten einzustufen.

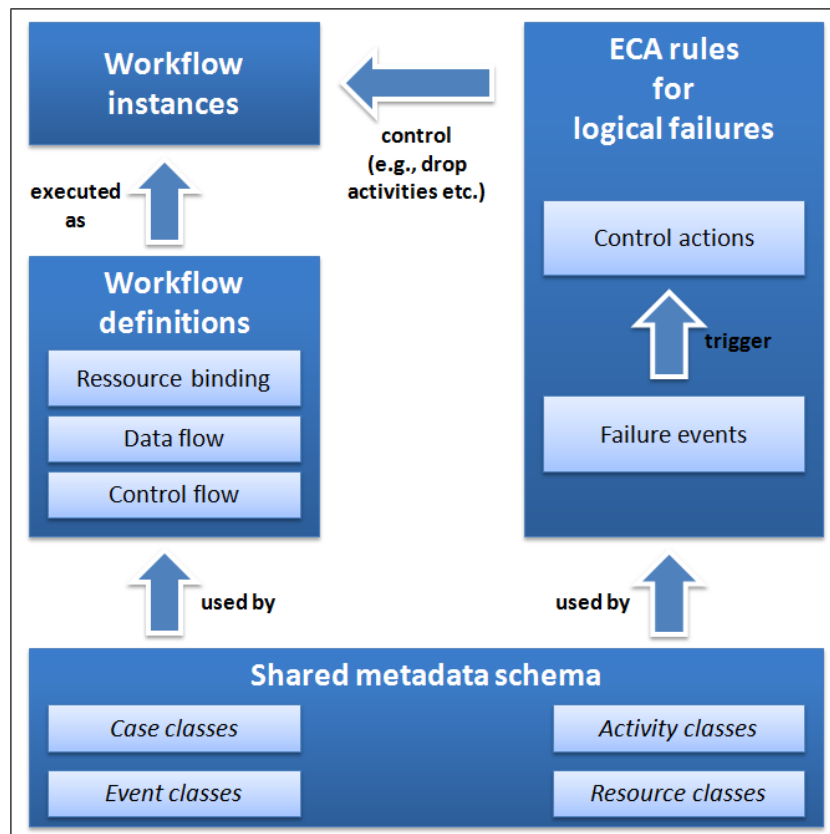


Abbildung 3.3: Integration von ECA in WFMS [85, S. 5] .

3.2.5 Controlflow-Constraints

Grundlagen und Entstehung des Modells

In diesem Abschnitt wird die Möglichkeit analysiert, auf die von Dr. Shazia W. Sadiq [59] vorgeschlagenen Kontrollflussbedingungen („controlflow-constraints“) im Rahmen des Konzeptes zu nutzen. Die Motivation für die Entwicklung dieses Modells ergibt sich aus der Beobachtung, dass die in heutigen betrieblichen Umgebungen zum Einsatz kommenden Workflowmanagementsysteme auf komplexe, vorhersagbare und sich oft wiederholende Prozesse ausgelegt sind. Dies bedeutet jedoch, dass die Prozessketten immer gleich aufgebaut sind und alle möglichen Abweichungen im Vorfeld ermittelt, erkannt, definiert und als Varianten modelliert werden müssen. Zur Laufzeit wird, abhängig von der gegenwärtigen Situation, die passende Variante ausgewählt und der Prozess entsprechend bearbeitet.

Um einen höheren Grad der Flexibilität bei der Modellierung und der Ausführung zu ermöglichen, wird folgendes Workflowmodell vorgeschlagen: Als Grundlage dient ein Kernprozess, der aus identifizierbaren und vordefinierten Workflowaktivitäten und Kontrollabhängigkeiten besteht. Zur Schaffung der Flexibilität werden bestimmte Flexibilitäts-Container („pockets

of flexibility“) definiert. Hierbei handelt es sich um spezielle Workflow-Aktivitäten („build activities“). Diese bestehen zum einen aus einer Menge von Workflow-Fragmenten, die entweder einzelne Aktivitäten oder ganze Sub-Prozesse darstellen können, und zum anderen einer Menge von Bedingungen („constraints“), die die Korrektheit innerhalb eines Flexibilitäts-Containers sicherstellen. In einem solchen Flexibilitäts-Container werden alle möglichen Bauteile deponiert und Regeln definiert, wie diese Bauteile miteinander verknüpft werden dürfen.

Workflow-Instanzen

Eine Instanz stellt einen in der Ausführung befindlichen Workflow dar und besteht initial nur aus dem Kernprozess und den Flexibilitäts-Containern. Die Instanz-Spezifikation ist entweder eine offene Instanz (zur Modellierungszeit) oder eine Instanzvorlage (zur Laufzeit). Eine Instanz ist somit eine konkrete, gültige Ausprägung eines flexibel gestalteten Workflows.

Kontrollflussbedingungen

Es werden fünf verschiedene Arten von Bedingungen definiert:

1. Serial
Alle Elemente müssen ausgeführt werden, wobei die Reihenfolge der Ausführung irrelevant ist.
2. Order
Hier wird lediglich die Reihenfolge festgelegt, in der die einzelnen Elemente abgeschlossen sein müssen. Die Ausführung muss nicht notwendiger Weise unmittelbar nacheinander erfolgen, es können Schritte dazwischen eingefügt werden.
3. Fork
Hier muss eine bestimmte Teilmenge aus einer vorher definierten Menge von Aktionen ausgeführt werden. Es darf kein Pfad zwischen den Elementen existieren. Ein Beispiel wäre, dass aus einem Pool von k möglichen Veranstaltungen, n ($n < k$) belegt werden müssen.
4. Include
Die Include-Bedingung (Include [a, {d,r,g}]) zu einer Aktion legt fest, dass, wenn diese eine bestimmte Aktion (a) in einen Pfad eingefügt wird, auch eine (oder mehrere) bestimmte weitere Aktionen ausgeführt werden müssen. (Ein Beispiel wäre, dass eine Hotelbuchung einen Bustransfer beinhaltet.)
5. Exclude
Wenn eine bestimmte Aktion gewählt wurde, darf eine (oder mehrere) andere nicht mehr hinzugefügt, bzw. ausgeführt werden. Z. B. schließt eine durchgeführte Barzahlung alle anderen Zahlungsarten (EC, Kreditkarte, Rechnung, etc.) aus.

Konflikte

Zwei oder mehr Constraints stehen in einem Konflikt, wenn die Kombination der beiden es verhindert, dass mindestens ein Fragment nicht mehr eingefügt werden kann, ohne dass ein Template ungültig wird. Ungültig bedeutet in diesem Fall, dass eine oder mehr der o. g. Regeln nicht eingehalten werden können. Beispielsweise stünden die Regeln „Include (A, {B, C})“ und „Exclude (C, {A})“ in einem Konflikt. Die erste Regel besagt, dass „Wenn Element A vorkommt, müssen auch die Elemente B und C enthalten sein, während die zweite Regel Element A ausschließt, sobald Element C enthalten ist. Somit wird es unmöglich beide Regeln gleichzeitig einzuhalten.“

Auflösung von Konflikten

Zunächst wird empfohlen, die Constraintmengen zu minimieren, um Übersichtlichkeit und Berechnungseffizienz zu ermöglichen. Jedoch basieren die Minimierungs-Ansätze auf NP-vollständigen Algorithmen. Somit ist abzuwägen, ob eine Minimierung sinnvoll ist.

Minimierung

- Order
Aus allen Order-Constraints werden gerichtete, azyklische Graphen erstellt, wobei die Knoten den Aktivitäten entsprechen, $A \rightarrow B$ entspricht A vor B = gerichtete Kante von A nach B. Die Menge der längsten Pfade ohne die doppelte Benutzung von Kanten entspricht der minimalen Menge der Order-Constraints.
- Serial
Aus allen Elementen der Serial-Constraints werden ungerichtete, vollständig verbundene Graphen erstellt, wobei die Knotenmenge der Menge der Aktivitäten entspricht. Die minimale Menge der Serial-Constraints entspricht der Menge der Cliques in dem Graphen. Hierbei ist das Cliques-Problem NP-vollständig.
- Include / Exclude
Die Include Regeln, z. B. $I(A, \{B, C\})$ werden verstanden als „Wenn A, dann B und C“ oder als logische Formel dargestellt: $A \Rightarrow B \wedge C$. Zu jedem Include Constraint wird eine Wahrheitstabelle erstellt. Alle Zeilen, die „false“ ergeben, werden gestrichen. Alle vorkommenden Zeilen müssen übereinstimmen. Wenn also eine Tabelle nur die eine Zeile $A=1, B=0$ aufweist, die in einer anderen Tabelle (über A und B) nicht enthalten ist, stehen diese beiden Regeln in Konflikt. Für die Exklusion ist die logische Formeln entsprechend aufzustellen: z.B. $E(A, \{B, C\})$, also „Wenn A, dann nicht B und nicht C“ ergibt $A \Rightarrow (\neg B \wedge \neg C)$. Die Verifikation über die Wahrheitstabellen erfolgt analog.

Verwendung für DynamicFlow

Ad-hoc-Modifikationen

Dieses Konzept ist dazu ausgelegt, zur Laufzeit Änderungen (in einem gewissen Umfang) am Prozessgraphen vorzunehmen, jedoch können Modifikationen streng genommen nur in den Flexibilitäts-Containern vorgenommen werden. Um den Grad der Flexibilität zu erhöhen ist es jedoch möglich, den gesamten Prozess als Flexibilitäts-Container zu definieren. Es bleibt jedoch das Problem, dass die Ad-hoc-Modifikationen darin bestehen, aus einem vordefinierten Pool aus Aktivitäten eine Teilmenge im Rahmen der Kontrollflussbedingungen auszuwählen. Eine komplett freie Modellierung zur Laufzeit ist somit nicht möglich.

Datenfluss

Dieses Konzept gibt keine direkten Vorgaben für die explizite Modellierung des Datenflusses, da es sich jedoch nicht um eine konkrete Workflow-Ausführungssprache handelt, kann das Konzept um Datenflussbedingungen erweitert werden, ist aber nicht Teil dieser Arbeit.

Sichten

Eine Modellierung aus verschiedenen Sichten ist nicht vorgesehen. Dies bedeutet, dass Workflows, die basierend auf diesem Konzept modelliert werden, nicht ohne tiefere IT-Kenntnisse von Fachanwendern aus informatikfernen Disziplinen erstellt oder bearbeitet werden können.

Vor- und Nachteile

Vorteile:

- verifizierbar
- es können nur gültige Lösungen modelliert werden
- klare Struktur

Nachteile

- begrenzte Flexibilität
- komplex
- keine Ad-hoc-Modifikationen
- Optimierung und Validierung basiert z.T. auf NP-vollst. Konzepten

3.2.6 Dataflow-Constraints

Nachdem im vorherigen Abschnitt (3.2.5) auf die Kontrollflussbedingungen eingegangen wurde, soll hier eine Möglichkeit vorgestellt werden, den Datenfluss zu modellieren. Als Grundlage dient ein weiterer Ansatz von Dr. Shazia W. Sadiq et. al. mit dem Titel Datenflussbedingungen („Dataflow-Constraints“) [58]. Das Konzept entstand aus der Erkenntnis, dass eine saubere Datenmodellierung über den gesamten Workflow hinweg notwendig ist. Es werden mehrere eventuelle Datenflussprobleme erläutert, die die korrekte Ausführung der Prozesse verhindern können. Ferner wird aufgezeigt, dass es zum Stillstand von Prozessen kommen kann, falls diese mit inkonsistenten Daten ausgeführt werden. Aus diesem Grunde zeigt das Konzept Möglichkeiten auf, Dateninkonsistenzen bereits während der Modellierung eines Workflows zu erkennen und aufzulösen. Schließlich werden Anforderungen an ein Workflow-Datenmodell entwickelt, die eine Datenvalidierung ermöglichen.

Grundlagen

Die folgende Liste zeigt die Hauptanforderungen an Datenfluss-Mechanismen innerhalb eines Workflow-Modells:

- **Datenverwaltung**
Innerhalb eines Workflow-Modells muss klar definiert sein, welche Daten von dem jeweiligen Arbeitsschritt benötigt werden, bzw. welche Daten ausgegeben werden.
- **Verfügbarkeit der erforderlichen Daten**
Es muss garantiert werden, dass die von einem Arbeitsschritt benötigten Daten (unter Umständen von anderen Arbeitsschritten) zur Verfügung gestellt werden.
- **Konsistenter Datenfluss**
Es müssen Mechanismen entwickelt werden, die einen konsistenten Datenfluss von einem Arbeitsschritt zu einem anderen Arbeitsschritt sicherstellen.

Formale Beschreibung ([58] S.3)

Sei *WF-Data* eine Prozessdatenbank (Process Data Store), die eine Menge von Datenelementen $\{V_1, V_2, \dots, V_k\}$ für einen Workflow W enthält. Der Workflow W wird definiert durch einen gerichteten Graphen, bestehend aus den endlichen Mengen N und F von Knoten bzw. von Kanten. Der Kontrollfluss F wird als Flussrelation auf $F \subseteq N \times N$ dargestellt. Die Menge der Knoten N wird in Tasks T und Controlflow-Constraints C unterteilt, mit $C \cap T = \emptyset$.

Es gilt:

$\forall n \in N$, $\text{Input}(n)$ ist die Menge von Datenelementen V_i^n aus $\{V_1, V_2, \dots, V_k\}$, die von n konsumiert werden.

$\forall n \in T$, $\text{Output}(n)$ ist die Menge von Datenelementen V_o^n aus $\{V_1, V_2, \dots, V_k\}$, die von n produziert werden.

Hier ist zu beachten, dass nur Knoten aus T Datenelemente generieren können, da Knoten aus C nur Daten lesen müssen, um Entscheidungen bei Kontrollflussbedingungen treffen zu können.

Datenmodell Anforderungen

Die Datenelemente an sich werden in folgende Typen unterteilt:

- Referenzdaten
Dies sind Datenelemente, die zur Identifikation einer Instanz oder eines Falles dienen, z. B. eine Patientenummer, Behandlungs-ID, etc.
- Operationelle Daten
Datenelemente, die für eine bestimmte Aktivität notwendig sind, z. B. die Anschrift eines Patienten oder eine konkrete Diagnose.
- Entscheidungsdaten
Diese Datenelemente stellen eine Teilmenge der operationellen Daten („Operational“) dar und sind notwendig, um Entscheidungen treffen zu können, z. B. das Patientenalter oder die Eigenschaften einer Diagnose.
- Kontextdaten
Hierbei handelt es sich um Datenelemente, die typischer Weise eine komplexere Struktur aufweisen und nicht unbedingt vordefiniert werden müssen. Solche Daten werden prinzipiell zugunsten von einzelnen Arbeitsschritten als Eingaben konsumiert sowie auch als Ausgaben produziert, wie z. B. ein Behandlungsplan oder ein Laborbericht.

Neben der Unterscheidung der verschiedenen Datentypen werden bei der Datenquelle zwischen „intern“ und „extern“ unterschieden, die wie folgt definiert sind:

- Interne Quelle
Von einer internen Quelle wird gesprochen, falls die Datenelemente in der Prozessdatenbank vorhanden sind, d.h. Lese- und Schreiboperationen erfolgen hier durch Wertzuweisungen.
- Externe Quelle
Hierbei handelt es sich um Daten, die von Fremdsystemen in den Workflow eingebracht werden. Dies könnten z. B. klinische Informationssysteme (siehe Abschnitt 2.4) oder ähnliches sein, wobei die Daten sowohl in die Prozessdatenbank übernommen, als auch lediglich durch einen lesenden Zugriff verwendet werden können.

Die Prozessdatenbank kann in diesem Kontext als eine Menge von Datenelementen oder als eine komplexere Struktur, mit sowohl komplexen Datentypen (z. B. Bilder), als auch unstrukturierten Daten (z. B. Textdokumente) definiert werden.

Datenvalidierung

Die Datenvalidierung dient dazu, Inkonsistenzen aufzulösen und Fehlersituationen durch falsche Daten zu vermeiden. Es wird zwischen folgenden Fehlersituationen bei der Datenvalidierung unterschieden:

- Redundante Daten
Hierbei handelt es sich um Daten, die zwar erzeugt und modelliert, jedoch an keiner weiteren Stelle im Workflow genutzt werden.
- Verlorene Daten
In diesem Kontext können Daten verloren gehen, falls zwei Arbeitsschritte, die parallel ausgeführt werden, ein und dasselbe Datum erzeugen und speichern. Hierbei wird das Ergebnis, welches der (zeitlich gesehen) frühere Speichervorgang gesichert hat, überschrieben. Es seien z. B. zwei Arbeitsschritte *A* und *B*, die parallel ausgeführt werden und separat ein Datum *X* generieren. Falls sich keine Kontrollabhängigkeit zwischen *A* und *B* befindet, könnte nicht mehr bestätigt werden, welcher Arbeitsschritt zuerst terminiert. Folglich könnte nicht mehr entschieden werden, ob ein darauffolgender Arbeitsschritt den Wert der Variablen *X* aus Arbeitsschritt *A* oder aus Arbeitsschritt *B* bekommt.
- Fehlende Daten
Es kann vorkommen, dass für einen bestimmten Arbeitsschritt ein Parameter als Eingabe gebraucht wird, jedoch weder eine Quelle, noch ein erzeugender Arbeitsschritt für den Parameter spezifiziert werden.
- Falsch zugeordnete Daten („mismatched-data“)
Dieser Fehler tritt auf, falls ein Arbeitsschritt einen bestimmten Verbund-Datentyp erwartet, die empfangenen Daten jedoch nicht dieselben (passenden) Datenfelder aufweisen, z. B. unterschiedliche Komponenten im Datentyp „Adresskopf“.
- Inkonsistente Daten
Wenn ein Arbeitsschritt ein Datum V_k aus einer Anwendung oder von einer interagierenden Person bekommt, speichert er das Datum und den zugehörigen Wert in eine Prozessdatenbank. Es wäre möglich, dass dieses Datum und dessen Wert später durch einen anderen Arbeitsschritt verwendet bzw. gelesen werden. Während dem Idle-Time (die Zeit, in der der Arbeitsschritt das Datum aus der Datenbank holt und bevor er dieses liest.) könnte eine externe Aktualisierung des Wertes zu einer „inkonsistenten Darstellung“ des Datums führen. Aufgrund dieses Problems wäre es unmittelbar nötig an eine Aktualisierungsstrategie der Daten innerhalb der Datenbank zu denken, die es ermöglicht solche Inkonsistenzen zu vermeiden.
- Fehlgeleitete Daten
Daten- und Kontrollfluss müssen zusammen passen, d.h. ein Arbeitsschritt muss zeitlich betrachtet Daten erzeugen, bevor ein anderer Arbeitsschritt diese Daten ver-

braucht. Hier wurden die benötigte Daten zwar spezifiziert aber nicht rechtzeitig von einem Arbeitsschritt zu einem anderen weitergeleitet.

Implementierungsmodelle für Datenflussmodellierung

- Im Falle des expliziten Datenfluss werden Datentransferoperationen als Teil des Workflowmodells definiert, wie in Abbildung 3.4 dargestellt. Hier fließen Daten von Arbeitsschritt A zu Arbeitsschritt C, wodurch sich eine entsprechende Datenabhängigkeit ergibt. Wenn Arbeitsschritt A beendet wird, werden die benötigten Eingangsdaten für Arbeitsschritt C erzeugt und entsprechend abgelegt. Arbeitsschritt B ist nicht von dieser Datenabhängigkeit betroffen.

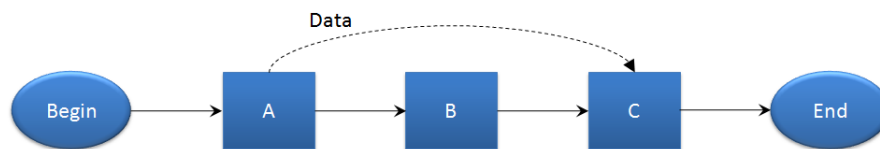


Abbildung 3.4: Expliziter Datenfluss [58].

- In Abbildung 3.5 wird der implizite Datenfluss dargestellt. Hierbei fließen die Daten mit den Kontrollflussinformationen. Der Nachteil hierbei ist, dass auch bei den Arbeitsschritten Datenströme auftreten, die durch unbenötigte Daten verursacht werden. Hier fließen z. B. die Daten, die von Arbeitsschritt A erzeugt und von Arbeitsschritt C benötigt werden über Arbeitsschritt B durch zwei Kontrollflüsse, nämlich den zwischen A und B und den zwischen B und C.



Abbildung 3.5: Impliziter Datenfluss über den Kontrollfluss [58].

- Impliziter Datenfluss durch eine Datenbank („Repository“). Die Prozessdatenbank stellt ein solches Repository dar. Hierbei senden die einzelnen Arbeitsschritte die erzeugten Daten an die Datenbank, worauf die verbrauchenden Arbeitsschritte durch Leseoperationen zugreifen, wie in Abbildung 3.6 dargestellt.

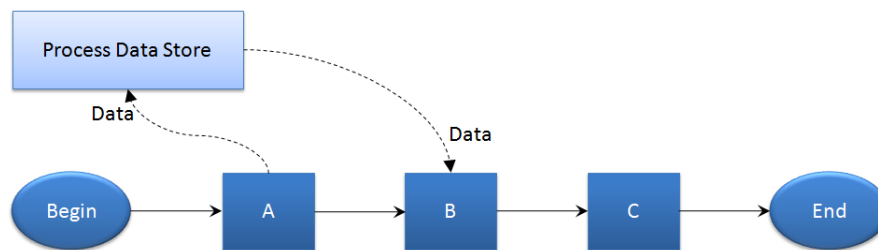


Abbildung 3.6: Impliziter Datenfluss über ein „Repository“ [58].

Verwendung für DynamicFlow

Je nach realisierter Art des Datenflusses ist dieses Konzept dazu geeignet, Ad-hoc-Modifikationen zu unterstützen. Gerade der implizite Datenfluss über eine Datenbank eignet sich hervorragend bei Ad-hoc-Modifikationen, da hier nur sichergestellt sein muss, dass benötigte Daten auch zur Verfügung gestellt werden. Beispielsweise darf kein Arbeitsschritt entfernt werden, der Daten erzeugt, die von einem anderen verwendet werden. Eine mögliche Lösung wären hier Meldungen an den Anwender, dass die Arbeitsschritte, die entsprechende Daten nutzen auch entfernt werden müssen. Auf die anderen Aspekte, wie verschiedene Sichten oder Vor- und Nachteile in Bezug auf die mögliche Verwendung hat die Analyse dieses Konzepts keine Auswirkungen.

3.2.7 Workflow-Patterns nach v.d. Aalst

Der Text *Workflow control-flow patterns - A Revised View* [55] von Russel et al. beschreibt verschiedene Pattern in Workflowsystemen. Unter Pattern werden hierbei häufig auftretende Workflow-Artefakte verstanden. Anhand dieser Beschreibungen evaluierte die Gruppe vorhandene Workflow Produkte. Untersucht wurde dabei, inwieweit die einzelnen Produkte die Umsetzung der beschriebenen Artefakte unterstützen.

Verstehen Gamma et al. ein Pattern primär als einen Lösungsansatz für wiederkehrende Probleme, werden Pattern in diesem Text in erster Linie als Beschreibungen für existierende Phänomene oder Konstellationen verwendet. Die Autoren verstehen ihr Werk wie folgt

„The original set of twenty patterns in the control-flow perspective were derived from a detailed examination of contemporary workflow systems and business process modelling notations in order to identify generic, recurring constructs. They have proven to be extremely popular with both theoreticians and practitioners alike as they provide a useful basis for discussing and comparing the capabilities and expressiveness of individual offerings in a manner which is independent of specific modelling formalisms and implementation technologies.”

Dieselbe Gruppe hat auch Pattern aus den Bereichen *Workflow Data Patterns* [53] und *Workflow Resource Patterns* [54] gesammelt. Alle drei Texte katalogisieren und untersuchen die gefundenen Pattern vor allem im Hinblick auf die Produktevaluierung.

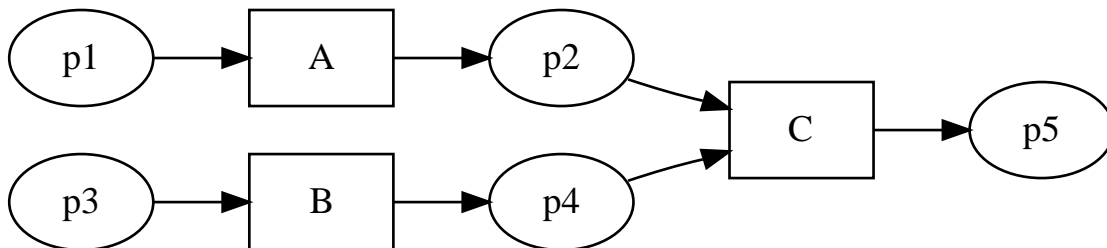
Control Flow Pattern

Control Flow Pattern beschäftigen sich mit dem Problem des Kontrollflusses in Workflows, primär also mit dem Teilen, dem Zusammenführen und dem Strukturieren von Kontrollflüssen. Die Beschreibung der Pattern erfolgt im Text „*Workflow control-flow patterns - A Revised View*“ [55] unter Zuhilfenahme vom Colored Petri-Nets. Hier werden zwei der Pattern exemplarisch beschrieben, die Definitionen im Text selbst sind jedoch deutlich umfangreicher.

Der Blickwinkel der Pattern ist die Evaluierung und eventuelle Entwicklung von (neuen) Produkten. Aus diesem Grund ist z. B. die Lösung eines Patterns als mögliches Vorgehen bei der Implementierung einer Workflow-Engine zu sehen und nicht als Lösung beim Design von Workflows.

Beide Beispiele sind mit Petri-Netzen dokumentiert. Dabei handelt es sich nicht um die Colored-Petri-Nets aus dem Text, sondern um analoge Stellen/Transitionen Netze.

Pattern WCP-3 – Synchronisation



Name *WCP-3 Synchronization*

Synonym *AND-join, rendezvous, synchronizer*

Zweck Zwei oder mehr Ausführungszweige werden derart in einen anschließenden Zweig vereinigt, dass der anschließende Zweig erst ausgeführt wird, wenn alle Eingabezweige ausgeführt wurden. Im Beispiel wird **C** erst dann ausgeführt, wenn die Transitionen **A** und **B** geschaltet haben.

Beteiligte Rollen Mehrere Eingabezweige, ein anschließender Zweig.

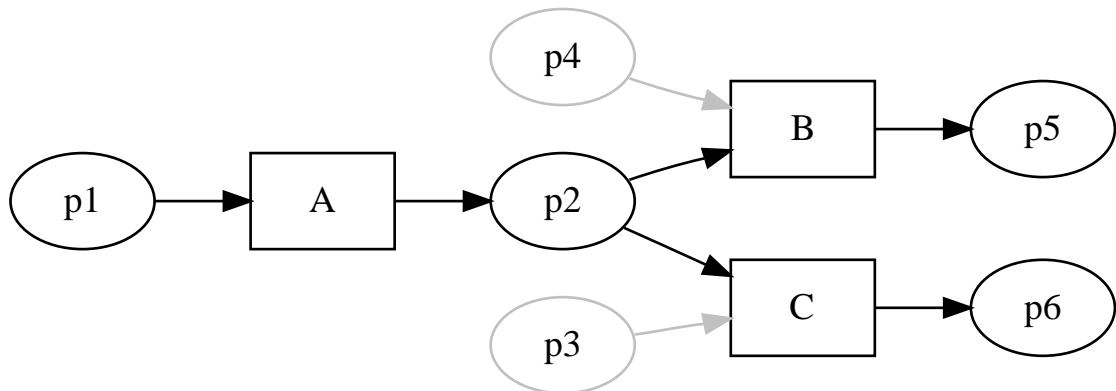
Kontext Dieses Pattern kann- bzw. soll angewandt werden, wenn jede der eingehenden Ausführungszweige genau einmal ausgeführt werden soll. Wird einer der Zweige mehrfach ausgeführt, so ist das Verhalten für dieses Pattern nicht definiert¹. Wird einer der Zweige niemals ausgeführt, so führt dies zu einem Deadlock in der *Synchronisierung*.

Lösung Jeder der eingehenden Zweige muss einmal ausgeführt worden sein, um den Ablauf fortzusetzen. Dies kann einmal explizit erfolgen, indem eine *AND-join* primitive eingeführt wird, oder implizit, indem eine Aktivität mehrere (bedingungslose) eingehende Kanten hat.

Beispiel Die Tagesabrechnung kann erst durchgeführt werden, wenn das Geschäft geschlossen wurde und die Kreditkartenbelege ausgedruckt sind.

Weitere Pattern Die eingehenden Pfade werden oft durch einen *AND-split* erzeugt.

Pattern WCP-16 – Deferred Choice



Name WCP-16 Deferred Choice

Synonym BPEL Pick, External choice, implicit choice, deferred XOR-split.

Zweck Es existieren für einen Prozess mehrere mögliche nächste Zweige. Der nächste auszuführende Ausführungszweig wird durch ein externes Ereignis bestimmt. Dieses Ereignis kann z. B. eine Nachricht oder ein Zeitgeber sein. Bis eines der Ereignisse eintritt, wartet der Prozess. Sobald eines der Ereignisse eingetreten ist, werden die anderen Ereignisse ignoriert.

Im Beispiel wird entweder **B** oder **C** – nicht jedoch beide – ausgeführt. Die Entscheidung wird im Zustand **p2** gefällt. Die Stellen **p3** und **p4** symbolisieren das externe Ereignis.

¹Die Semantik der Petrinetze ist schon definiert. Für dieses Pattern wird eine zweite Ausführung von der Betrachtung ausgeschlossen.

Beteiligte Rollen Mehrere mögliche nächste Ausführungszweige. Eventuell zeitgesteuerte Auslöser.

Kontext Hängt das weitere Vorgehen des Prozesses von einem von mehreren externen Ereignissen ab, so kann *Deferred Choice* verwendet werden, um auf das erste auftretende Ereignis zu reagieren.

Lösung Dieses Muster ist sehr komplex und wird zur Zeit nur von Token-basierten Workflow-Produkten implementiert.

Beispiel Wird eine Instanz des Workflows *Reklamation* erstellt, gibt es zwei weitere Ausführungszweige: *Kontakt mit dem Kunden aufnehmen* und *Fall eskalieren*. Die Aktivität *Fall eskalieren* wird automatisch aufgerufen, wenn nach 5 Werktagen kein Kontakt mit dem Kunden aufgenommen wurde.

Data Pattern

Data Pattern versuchen zu erfassen, wie Daten in Workflows fließen und das Verhalten der Workflows beeinflussen können. In *Workflow Data Patterns* [53] beschreiben Russel et al. verschiedene Pattern aus diesem Bereich. Wie auch in [55] geht es primär um die Katalogisierung von gefundenen Pattern zwecks Evaluierung von Software.

Laut Russel et al. weisen ein Großteil der gefundenen Pattern Eigenschaften auf, die sie in eine der folgenden vier Gruppen einteilen.

Sichtbarkeit der Daten Wie und in welchem Umfang können die verschiedenen Komponenten des Workflows auf Daten zugreifen?

Interaktion Wie können Daten zwischen den einzelnen Komponenten ausgetauscht werden, welche Kommunikationswege existieren?

Datentransfer Wie werden die Daten übergeben? (Stichwort: Wertübergabe gegenüber Referenzübergabe)

Steuerung durch Daten Wie können Daten den Workflow steuern oder beeinflussen?

Ein Pattern aus der Gruppe *Steuerung durch Daten* ist das *Data-based Task Trigger* Pattern. Es beschreibt, wie ein bestimmter Task des Workflows automatisch angestoßen wird, wenn eine bestimmte Bedingung sich erfüllt.

Copy By Value ist ein *Datentransfer* Pattern, bei dem den einzelnen Aktivitäten nur Kopien der Parameter übergeben werden.

Ressource Pattern

Die bis hierhin besprochenen Workflow-Pattern behandeln die Struktur und die Kommunikationswege von Workflows. Bei den in [54] vorgestellten Ressource-Pattern geht es primär darum, wie Aktivitäten verschiedenen Ressourcen zugewiesen werden. Dies geschieht durch eine aktive Zuweisung einer Aktivität an eine Ressource oder eine Ressource nimmt sich einer Aktivität an. In viele Fällen ist mit einer Ressource ein Mitarbeiter, eine Arbeitsgruppe oder eine bestimmte Mitarbeiterrolle gemeint. Hier gibt es jedoch keine begriffliche Einschränkung, so dass auch ohne weiteres eine spezielle Maschine oder ein Konferenzraum als Ressource gelten kann.

Das Pattern *R-CBA (Capability-based Allocation)* beschreibt, wie eine bestimmte Aktivität einer Resource aufgrund von bestimmten Fähigkeiten der Resource zugewiesen wird. Beispiel: „Das Angebot an die Air-France muss von einer französisch sprechenden Person aus dem Projekt *Black Box* verfasst werden“.

Eine rollenbasierte Zuweisung (*R-RBA - Role-Based Allocation*) an eine Resource ist die Einschränkung „Das Angebot muss von einer Person mit der Rolle *Manager* freigegeben werden“.

Flexibilität

Die vorgestellten Pattern wurden ermittelt, um Workflowsysteme zu evaluieren. Die dafür gewählte Form einer Aufzählung oder eines Pattern-Kataloges ist für diesen Zweck geeignet. Der für die PG relevante Aspekt der ad hoc Modellierung wird in den vorgestellten Pattern nicht angesprochen. Die vorgestellten (Kontrollfluss-) Pattern bieten sich jedoch an, um auf ihre Tauglichkeit in Bezug auf Ad-hoc-Modifikationen untersucht zu werden.

3.2.8 Umsetzung klinischer Pfade (Meiler)

Meiler entwickelt in seinem Buch[23] ein Konzept zur Modellierung klinischer Pfade in WfMS. Dazu betrachtet er klinische Pfade unter folgenden Aspekten: organisatorisch, betriebswirtschaftlich und evidenzbasiert.

Aus den einzelnen Gesichtspunkten (oder auch Aspekten) ergeben sich verschiedene Anforderungen an ein Modellierungskonzept (siehe Abbildung 3.7). Anschließend betrachtet er verschiedene Modellierungsansätze [23, S. 26ff] und kommt zu dem Schluss, zunächst ein Basiskonzept zu erstellen und dieses domänenspezifisch zu erweitern.

Zunächst identifiziert er dazu drei Sprachebenen für WfMS. An oberster Stelle steht dabei das Pfadmetametamodell, das quasi selbstbeschreibend die grundlegenden Modellierungskonstrukte festlegt. Das Pfadmetamodell ist eine Ableitung aus dem Pfadmetametamodell das die grundlegenden Konstrukte näher beschreibt und ihre Kombinationsmöglichkeiten festlegt. Das Pfadmodell ist eine modellierte Beschreibung eines klinischen Pfades, also

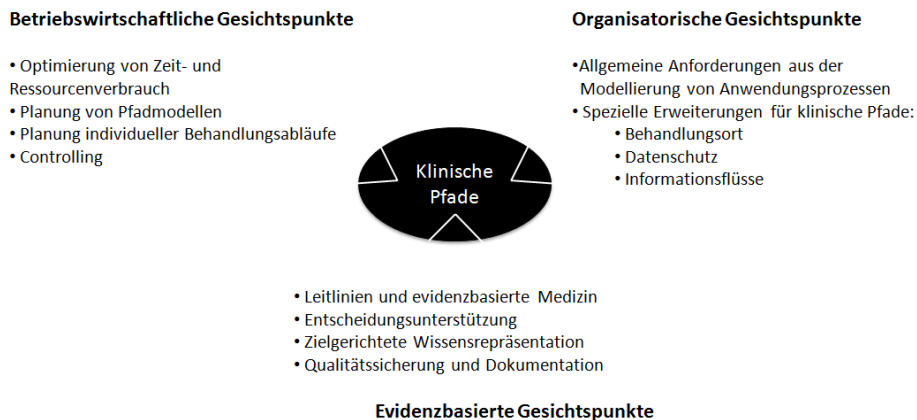


Abbildung 3.7: Aspekte Klinischer Pfade [23, S. 42] .

eine Beschreibung, die sich aus dem Metamodell ableiten lässt. Das Pfadmodell ist unabhängig von jedwedem Ausführungskontext. Im Kontext klinischer Pfade bedeutet das, dass ein Pfadmodell nicht den Pfad eines Patienten festlegt, sondern allgemeiner, welche Pfade ein Patient in einer Pfadinstanz durchlaufen kann. [23, S. 49f]

Beispielhaft könnte man sich das wie folgt vorstellen: Ein Pfadmetamodell würde festlegen, dass es Menschen, Informationen und Handlungen gibt. Das Pfadmetamodell legt fest, wie Menschen mit einer Handlung (sprechen / schreiben) Informationen austauschen. Ein Pfadmodell könnte z. B. beschreiben, wie Vorstellungssituationen in verschiedenen Kulturen ausgeführt werden. Die Pfadinstanz beschreibt den Ablauf den zwei Personen auf diesem „Vorstellungsplan“ durchlaufen.

Im vierten Kapitel seiner Monographie definiert Meiler abstrakte Prozessbausteine und stellt verschiedene Konzepte dar, wie diese Bausteine in einen Workflow integriert werden können: interne Verwendung, externe Verwendung, sowie Attributierung und wie diese durch die Verwendung von Flusstypen miteinander verbunden werden können. [23, S. 67ff]

Besonders interessant sind hier die verschiedenen Arten wie die Bausteine eingebunden werden können und die daraus resultierenden Vor- und Nachteile. Externe Verwendung bezeichnet die Vorgehensweise, dass Bausteine an einer Stelle definiert und als Verweis auf diese Definition in den Workflow eingebaut werden. Bei interner Verwendung hingegen wird eine Kopie des Bausteins in den Workflow „kopiert“. Bei der Attributierung wird ein Baustein kopiert, aber sein Inhalt kann noch durch einzustellende Parameter beeinflusst werden.

Interne und externe Verwendung kann mit der Verwendung symbolischer Links in Linux/Unix-Dateisystemen verglichen werden. Interne Verwendung wäre das Erstellen der Kopie einer Datei und externe Verwendung das anlegen eines symbolischen Links, der auf die Originaldatei verweist. Die Benutzung des Konzepts „externer Verwendung“ macht weitere Überlegungen über Vorgehensweisen bei Änderungen an Bausteinen notwendig. [23, S. 76ff]

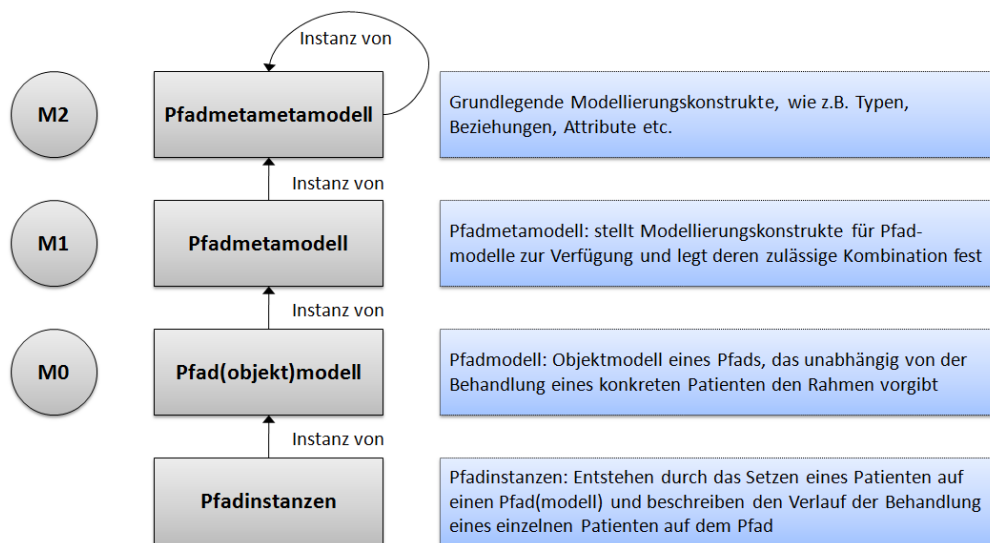


Abbildung 3.8: Modellhierarchie [23, S. 50] .

In den weiteren Kapiteln identifiziert Meiler Konstrukte für sein Pfadmetamodell und entwickelt Strategien, wie diese in das Gesamtkonzept eingepasst werden können.

Evidenzbasierter Aspekt:

Unter dem Aspekt der Evidenzbasierung identifiziert Meiler folgende Konzeptbausteine:

- **Evidenzbasierter Entscheider**[23, S. 98ff] ... als Möglichkeit mehrere Entscheidungen in einer zusammenzufassen. Auch können mit Hilfe dieses Konstruktes bestimmte Entscheidungen unter evidenzbasierten Gesichtspunkten vorgeschlagen werden.[23, S. 142ff]
- **Checkliste**[23, S. 106ff] ... als Möglichkeit unter Qualitätssicherungsgesichtspunkten menschliche Handlungen zu beschreiben.[23, S. 143]
- **Eskalationsmechanismus**[23, S. 114ff] ... erlauben die Ausführung von Aktionen zur „Ausnahmebehandlung“[23, S. 143]
- **Einbettung medizinischer Kontextinformation** [23, S. 126ff] ... ermöglicht es, wichtige, evidenzbasierte Kontextinformationen in den Prozessablauf zu integrieren. [23, S. 143]
- **Wiederholte Ausführung von Pfadelementen**[23, S. 131ff] Schleifenkonstrukte ermöglichen es, Pfadelemente abhängig von med. Bedingungen wiederholt durchzuführen. [23, S. 143]

- **Temporale evidenzbasierte Abhängigkeiten** [23, S. 135ff] ... erweitern Kontrollflusskonstrukte um die Möglichkeit zeitliche Zusammenhänge zu modellieren [23, S. 143]

Organisatorischer Aspekt:

- **Behandlungsort**[23, S. 147]
... ermöglicht die Integration der Umgebung in das Modell [23, S. 178]
- **Funktionszuordnung**[23, S. 152]
... trägt dem Punkt Rechnung, dass Personen innerhalb eines Pfades bestimmte Funktionen einnehmen können.[23, S. 178]
- **Spezifikation von Vertraulichkeit**[23, S. 158]
... ermöglicht es, bestimmte Informationen nur bestimmten Personenkreisen zur Verfügung zu stellen. Dies trägt vor allem Bedingungen des Datenschutzes Rechnung.
- **(Daten-)Filter an Organisationsgrenzen**[23, S. 163]
... (s.o. Allerdings mit dem Anspruch der Erweiterung des Pfades auf mehrere Organisationen)[23, S. 178]
- **Modellierung komplexer Datenelemente**[23, S. 167]
... ermöglicht es, mehrere einfache Daten zu komplexeren Elementen zusammenzufassen. [23, S. 178f] Man könnte sich hier z. B. vorstellen, dass verschiedene Blutwerte als einfache Datenelemente zum Datenelement „großes (oder kleines) Blutbild“ zusammengefasst werden.
- **Propagieren von Datenelementen**[23, S. 172]
... trägt der Forderung Rechnung, dass bestimmte Daten in bestimmter Reihenfolge erhoben werden müssen. An diesem Punkt werden die Anforderungen an den Datenfluss modelliert.[23, S. 179]

Betriebswirtschaftlicher Aspekt:

- **Klassifikation von Pfaddurchläufen** [23, S. 184]
... zur Erfassung von „häufigeren“ Pfadvarianten. Dient auch möglicher Optimierung.
- **Anreicherung des Pfadmodells um statistische Werte** [23, S. 172]
... zur erfassung von „häufigeren“ Pfadvarianten und Entscheidungen. Dient auch möglicher Optimierung.

- **Zeitplanung**[23, S. 172]
... Aufnahme von Zeitinformationen in das Pfadmodell
- **Kostenplanung**[23, S. 172]
... Annotation von Kosteninformationen an den Pfad.

Ad-hoc-ModifikationAspekt: Meiler stellt in seiner Monographie ein Modell zur Umsetzung Klinischer Pfade und der dazu benötigter Elemente vor. Eine Konzeptionelle Unterstützung von Ad-hoc-Modifikationist in diesem Konzept nicht vorgesehen. Die Arbeit bietet demnach eine Sinnvolle Basis zur Identifizierung von Elementen, muss aber noch Konzeptionell um eine Unterstützung von Ad-hoc-Modifikationerweitert werden.

Allgemeine Besonderheiten: Die einzelnen Aspekte werden in Pfadelementen umgesetzt, oder häufig auch an die Elemente bzw. an den Pfad annotiert.

Vorgriff auf Bewertung:

- Es besteht die Möglichkeit, die vorgestellten Aspekte im DynamicFlow-Konzept als Elemente in Aktionsplänen umzusetzen.
- Die Kapselung führt dazu, dass die Ausführungssemantik in die Bausteine gebracht wird, d.h.
 - Die Wiederverwendbarkeit wird erhöht.
 - Die Nutzung durch Fachanwender aus informatikfernen Disziplinen wird ermöglicht, da die Ausführungssemantik (transparent) im Baustein verankert ist.
- Folge: Die Konzepte bieten Einsatzmöglichkeiten für Ad-hoc-Modifikationen, weisen allerdings eine andere Organisation auf, um eine bessere domänenspezifische Verständlichkeit zu bieten.

3.3 Resultierendes Konzept

Bei der Analyse der existierenden Ansätze im Bereich von WfMS (siehe Kap. 3.2) stellte sich heraus, dass die mathematischen Konzepte wie z. B. Petri-Netze zwar sehr mächtig sind, dass jedoch nicht zu erwarten ist, dass Mediziner diese Art der Prozessmodellierung auf Anhieb beherrschen. Um den Anwendern den Einarbeitungsaufwand an dieser Stelle zu ersparen und um zu verhindern, dass die Anwendung aufgrund der hohen Komplexität ungenutzt bleibt, wurden weitere Konzepte untersucht. Hierbei wurde der Fokus verstärkt auf Anwendungen gelegt, die dafür ausgelegt sind, von Anwendern aus informatikfernen Disziplinen genutzt zu werden. Beispielfhaft seien hier ADEPT oder AgentWork genannt.

Leider stellte sich bei genauerer Betrachtung dieser Konzepte heraus, dass diese in der vorliegenden Form nicht alle Anforderungen, wie z. B. Ausführungssemantiken oder die Unterstützung von Ad-hoc-Modifikationen abdecken.

Aus diesem Grunde wurde ein neues Konzept erdacht, das zum einen Denkmodelle aus vorliegenden Arbeiten nutzt, zum anderen aber auch neue Elemente enthält. Sowohl die Einbindung der existierenden Komponenten oder Denkweisen, als auch die Vorstellung der neu entwickelten Konstrukte erfolgt im folgenden Kapitel.

3.3.1 Grundsätzliches Konzept

Der Grundbaustein dieses Konzeptes ist ein Aktionsplan, dieser ist gemäß der Abbildung 3.9 aufgebaut. Es handelt sich dabei um eine Art Container für einzelne Behandlungsschritte (im Folgenden als Instruktionen bezeichnet) und Bedingungen, die zur Ausführung dieser Schritte erfüllt sein müssen. Des Weiteren können (optional) Überwachungsregeln definiert werden, die festlegen, was in bestimmten Fehlersituationen unternommen werden soll. Demnach ergibt sich ein hierarchischer Aufbau von Aktionsplänen. Eine detaillierte Beschreibung der einzelnen Komponenten findet sich in späteren Abschnitten (siehe Kapitel 3.3.2, 3.3.4 und 3.3.5).

3.3.1.1 Akteure

Um den Einsatz im Klinikalltag zu ermöglichen, muss das Produktivsystem einfach zu nutzen sein und die oben genannten Designziele erfüllen (siehe Kapitel 3.1.1). Hierbei ist es erforderlich, dass verschiedene Benutzergruppen mit dem System interagieren, welche sich in die drei folgenden Spalten einteilen lassen:

1. Klinikpersonal (Endanwender)
2. fachliche Entwickler (Prozessmodellierer)
3. technische Entwickler (Programmierer)

Abbildung 3.10 gibt einen Überblick über die Interaktionen zwischen den einzelnen Nutzergruppen. Technische Entwickler erstellen Konstrukte, die von fachlichen Entwicklern zu Standardworkflows angereichert und kombiniert werden. Die letztliche Nutzung erfolgt durch das Ausführen dieser Workflows.

Endanwender sind die Mitarbeiter im Klinikalltag, hierbei handelt es sich um Ärzte, Pflege- und Verwaltungspersonal. Das System muss so gestaltet sein, dass es ohne größeren Einarbeitungsaufwand durch das Klinikpersonal genutzt werden kann. Dem Endanwender werden verschiedene Arbeitsabläufe (Workflows) zur Verfügung gestellt, die so vormodelliert wurden, dass die jeweilige Standardbehandlung abgedeckt ist. Bei der Ausführung können zwei Sonderfälle eintreten:

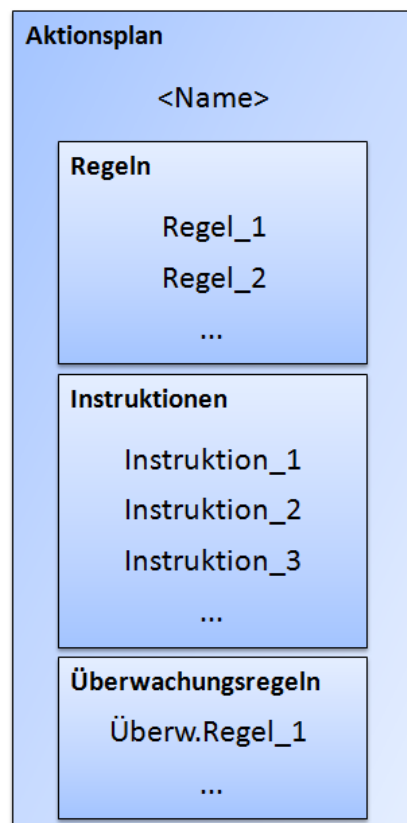


Abbildung 3.9: Aufbauschema eines Aktionsplans.

1. Es müssen Entscheidungen getroffen werden.
2. Es müssen Änderungen vorgenommen werden.

Die Reaktionen auf die Sonderfälle werden in diesem Konzept vom Klinikpersonal vorgenommen. Entscheidungen werden durch entsprechende Dialoge abgefragt, im Fall von Änderungen am modellierten Ablauf, stehen dem Anwender verschiedene Werkzeuge zur Verfügung, auf die in späteren Kapiteln genauer eingegangen wird.

Die Modellierung der Workflows wird von fachlichen Entwicklern durchgeführt, die über verschiedene Kompetenzen verfügen. Zum einen müssen sie medizinisch geschult sein, um die Behandlungsabläufe zu erkennen und zu optimieren, zum anderen müssen sie in der Lage sein, diese als Prozessmodell abbilden zu können. Den fachlichen Entwicklern wird eine Art Baukastensystem zur Verfügung gestellt, mit dessen Hilfe Standardabläufe modelliert werden können. Aus einzelnen Behandlungsschritten (Aktionsplänen) werden Workflows zusammengesetzt, indem technische Konstrukte mit medizinischem Fachwissen ergänzt

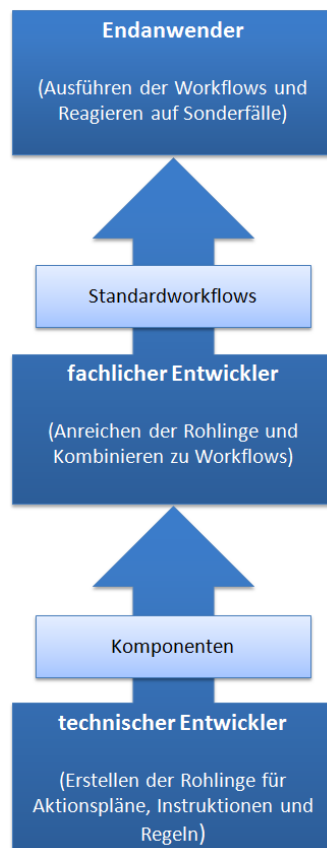


Abbildung 3.10: Interaktion der Nutzergruppen.

werden. Verzweigungen, im Sinne von Entscheidungen zwischen verschiedenen Behandlungsalternativen, werden ebenfalls durch das Einfügen von Regelbausteinen in die Aktionspläne realisiert.

Eine dritte Gruppe von Anwendern ist mit der Erstellung der technischen Konstrukte beauftragt. Diese technischen Entwickler erstellen mit Hilfe bestimmter Modellierungstools unter Verwendung geeigneter Sprachen zur Modellierung von Workflows komplexe Konstrukte, die von den fachlichen Entwicklern weiter verarbeitet werden.

3.3.1.2 Datenmodellierung

Motivation Die medizinischen Daten werden in diesem Ansatz als „vorhanden“ betrachtet. Oft existieren in den Kliniken bereits klinische Informationssysteme (KIS, siehe Kapitel 2.4) in denen die zur Behandlung erforderlichen Patienten-, Diagnose- und Verwaltungsdaten erfasst sind.

In diesem Kontext könnte das System so erweitert werden, dass bei der Ausführung des Workflows, die notwendigen Daten aus dem KIS bezogen werden. Ferner könnten auch

Labordatensysteme, Dokumentationsdatenbanken oder ähnliche Systeme angebunden werden, um Daten systemübergreifend verarbeiten zu können.

Mögliche Standards wären hier Health Level Seven (HL7) [10], Clinical Document Architecture (CDA) [11] oder Digital Imaging and Communications in Medicine (DICOM) [3] auf die hier aber nicht weiter eingegangen werden soll.

Beschreibung In diesem Ansatz werden daher alle notwendigen Daten aus einer abstrakten Schnittstelle eingebunden. Somit finden sie in den entsprechenden Komponenten ihre Verwendung, ohne dass explizite Datentransfermechanismen implementiert wurden.

Ebenfalls wird davon abgesehen, einen expliziten Datenfluss zu modellieren. Allerdings werden dem Modellierer alle erforderlichen Daten zur Verfügung gestellt, indem diese als Baustein aus einer Art Datenvorrat entnommen werden können. Eine Möglichkeit wäre, aus einer Liste den Baustein „Standard-Blutbild“ zu entnehmen und an entsprechender Stelle, z. B. in einer Regel abzulegen.

Erfassung von Metadaten Zusätzlich zu den medizinischen Daten ist es wichtig, Metadaten zu erfassen. Sei es aus Kontrollfluss- oder Dokumentationszwecken und somit auch Abrechnungs- und Optimierungszwecken.

Modellebene Zu jedem Element des Metamodells werden daher zusätzliche Informationen gespeichert. Diese richten sich z. B. nach den Ansprüchen, die an die Evaluation von durchlaufenen Pfaden oder Anwendungen von Elementen gestellt werden. Es bietet sich daher an, mindestens die folgenden zusätzlichen Informationen zu modellieren:

- **Beschreibung und Hilfe zur Anwendung eines Elements (bzw. Bausteins):**
Durch eine Beschreibung der Aufgabe und der Funktion eines Elementes erleichtert sich die Kommunikation zwischen den einzelnen Sichten der jeweiligen Bearbeiter. Gerade bei Ad-hoc-Modifikationen ist anzunehmen, dass eine bessere Dokumentation der Elemente ihre Wiederverwendbarkeit erhöht. Des Weiteren muss dem Modellierer, der den Baustein unter Umständen nicht entworfen hat, klar sein, was eine Verwendung des Bausteins für Auswirkungen hat (z.B. Versand einer Mail, Ausführen von Anmeldungen, Buchungen, etc.).
- **Name / Typ / Version**
Sind Informationen die für eine spätere Auswertung über Workflowgrenzen hinaus notwendig sind, um Aussagen der Form Instruktion A benötigt in etwa 3 Stunden, im Rahmen von Workflow B hingegen ca. 2 Stunden, etc.).

- Für das Qualitätsmanagement (QM) relevante Informationen, wie z. B. Erstellungsdatum oder Ersteller.
Dies bietet zum einen gewissen Schutz vor Missbrauch durch Protokollierung, zum anderen auch die Möglichkeit, Fragen zur Implementierung eines Bausteins direkt an den Ersteller zu stellen.
- Kontextinformationen
Dies kann hilfreich sein, um die Bedeutung eines Elements im Workflow besser einschätzen zu können. Es können z. B. bei Elementen aus medizinischen Leitlinien eine Erklärung, was dieses Element repräsentieren soll und einen Verweis auf die Leitlinie gespeichert werden. Hier spiegelt sich zum einen der evidenzbasierte Aspekt, zum anderen auch Teile des organisatorischen Aspekts wieder, indem hier Richtlinien, krankenhausinterne Regelungen, sowie Informationen zu Leitlinien hinterlegt sein können.
- Es könnten auch statistische Informationen aus vorhergegangenen Ausführungen als Hilfeleistung angeboten werden: z. B. die erwartete Ausführungszeit, auf Basis bisheriger Ausführungen, etc. Diese können dynamisch zur Laufzeit oder auch zwischengespeichert zur Verfügung gestellt werden, um Designentscheidung zu unterstützen. Zudem können hierdurch leichter kritische Pfade (Pfad der den längsten Workflowdurchlauf impliziert) identifiziert werden.

Auf Instanzebene Um in späteren Auswertungen die Elemente differenziert erfassen zu können, müssen Elemente des Workflows mit bestimmten zusätzlichen Informationen behaftet sein, um eine spätere Identifizierung in einem Protokoll möglich zu machen. Minimal sollten dazu folgende Informationen erfasst werden:

- ID für die Workflowinstanz
- ID/Name/Version des Elements
- ID/Name des Patienten
- Zeitpunkt der Aktivierung des Elements / Beendigung des Elements / Status der Beendigung
- ID der Ressourcen (Wer bzw. was war beteiligt?)

Umgebung (Erweiterbar)

- Aktuelle Uhrzeit
- Anzahl Workflows pro Patient
- Aufenthaltsort des Patienten

3.3.2 Instruktionen

Motivation

Anweisungen in gängigen Modellierungssprachen wie z. B. BPEL sind für den Anwender meist keine nachvollziehbaren Konstrukte. Von daher werden in diesem Konzept Anweisungen der Modellierungssprache zu Instruktionen gebündelt.

Beschreibung

Instruktionen verkörpern vorgefertigte Anweisungen, die vom Modellierer aus einem Vorrat entnommen werden können. Die einfache Bündelung von Anweisungen würde jedoch noch kein ausreichendes Maß an Flexibilität der Bausteine gewährleisten. Dieses lässt sich erreichen, indem Instruktionen so aufgebaut sind, dass sie Parameter, wie z. B. Daten entgegennehmen können und daraus gültige Anweisungen erstellt werden.

Um den fachlichen Entwickler, bei der Konfiguration des Elements zu unterstützen, wird eine grafische Schnittstelle angeboten, in der alle Parameter der Instruktion in einem Formular gesetzt werden können. Diese Schnittstelle wird entsprechend von den technischen Entwicklern erstellt. Mit ihrer Hilfe lassen sich neben den Instruktionen auch die restlichen Teile des Workflows konfigurieren.

Instruktionen sind in ihrer Art und Weise von den technischen Entwicklern vorgegeben. Innerhalb der Instruktion können Daten über die oben beschriebene Schnittstelle geschrieben, manipuliert und gelesen werden. Eine Möglichkeit wäre z. B. ein Röntgenbild an den behandelnden Hausarzt weiterzuleiten.

3.3.3 Instruktionsgruppen

Motivation

Oft lassen sich bestimmte Arbeiten in kleinere Schritte aufteilen, die in einer bestimmten Reihenfolge ausgeführt werden müssen. Instruktionsgruppen bilden eine einfache Möglichkeit Instruktionen zusammenzufassen.

Formalisierung

Instruktionsgruppen definieren eine Halbordnung auf Instruktionen und stellen sicher, dass die Instruktionen gemäß dieser Ordnung abgearbeitet werden. Instruktionsgruppen können geschachtelt definiert werden, demnach enthalten sie sowohl atomare Instruktionen, als auch vollständige Instruktionsgruppen.

Beschreibung

Instruktionsgruppen bestehen aus bestimmten Elementen, die Vorbedingungen und daran gekoppelte Instruktionen bzw. Instruktionsgruppen darstellen. Dabei wird eine Instruktion immer dann ausgeführt, wenn keine Vorbedingungen mehr zu erfüllen sind. Eine Vorbedingung für eine bestimmte Instruktion stellt in diesem Kontext eine Menge von Instruktionen bzw. Instruktionsgruppen dar, deren Fertigstellung für die aktuelle Instruktion vorausgesetzt wird.

3.3.4 Regeln

Motivation

Im medizinischen Umfeld ist es notwendig, bestimmte Behandlungsschritte von Vorbedingungen abhängig zu machen. Man denke z. B. an die Notwendigkeit, ein EKG zu erstellen, falls bei der Narkosevorbereitung festgestellt wird, dass der Patient eine kardiologische Vorerkrankung hat.

Aus diesem Grund bietet der Ansatz die Möglichkeit, die Ausführung von Aktionsplänen und Instruktionen von Vorbedingungen abhängig zu machen. Je nach Einsatzgebiet weisen die Regeln verschiedene Wirkungen auf. Somit werden folgenden Typen unterschieden:

- **Ablaufregeln:**
Bei Ablaufregeln handelt es sich um Bedingungen, die erfüllt sein müssen, um Instruktionen innerhalb von Aktionsplänen auszuführen, sie werden im Abschnitt [3.3.4.2](#) beschrieben.
- **Überwachungsregeln:**
Diese Form der Regeln sorgt für die Behandlung von Ausnahmesituationen innerhalb eines Aktionsplanes. Eine detaillierte Beschreibung findet sich in Kapitel [3.3.4.3](#).
- **Aufbauregeln:**
Im Gegensatz zu den ersten beiden Regeltypen handelt es sich bei Aufbauregeln um Konstrukte, die die Gültigkeit von verschiedenen Kombinationen von Aktionsplänen innerhalb eines Workflows festlegen. Sie werden in Kapitel [3.3.8.2](#) detailliert beschrieben.

3.3.4.1 Basisregeln

Formalisierung Eine Regel ist eine Relation auf Daten, die diese nach Wahr, Falsch, oder „nicht anwendbar“ (n.a.) abbildet.

Für zweistellige Funktionen sei diese wie folgt definiert:

$$R : D^n \times O \rightarrow \{\text{wahr}, \text{falsch}, \text{n.a.}\}, d \in \text{Daten} \wedge o \in \text{Operationen}$$

- R, Mender der erstellbaren Regeln
- D, Menge der Daten die in der Datenwolke zur Verfügung stehen.
- O, Menge der auf den Daten definierten Operationen.

Regeln sind Abbildungen von Daten, unter Berücksichtigung einer Operation, auf die Werte:

- True,
wenn die Daten unter der gegebenen Operation per Definition zu wahr ausgewertet werden.
- False,
wenn die Daten gemäß der Operation zu falsch ausgewertet werden.
- n.a.,
falls die benötigten Daten nicht Verfügbar sind.

Beschreibung Die Definition von Regeln erfolgt über einen Dialog, wie in Abbildung 3.11 dargestellt, so dass ein Modellierer auch ohne Programmierkenntnisse in der Lage ist, Regeln zu definieren. Hier werden sämtliche Parameter und Operatoren einer Sammlung entnommen und kombiniert. Die Auswertung erfolgt zu „wahr“ bzw. „falsch“ und so gewissermaßen als „Schalter“ für die Einhaltung der Ausführungssemantik (siehe Kap. 3.3.9).

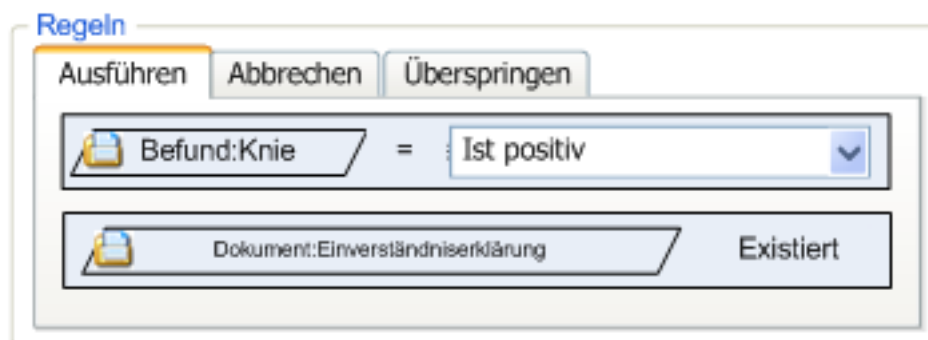


Abbildung 3.11: Beispiel für einen Regelbaustein im Editor.

3.3.4.2 Ablaufregeln

Motivation Bei der Auswertung von Ablaufregeln werden folgende drei Ziele unterstützt:

1. Überspringen von Instruktionen
Ein Anwendungsfall wäre, kein Röntgenbild bei der aktuellen Behandlung eines Patienten zu erstellen, wenn bereits eine äquivalente Aufnahme, z. B. beim Hausarzt vorliegt, die angefordert werden kann.

2. Warten auf Werte

Aus medizinischer und organisatorischer Sicht kann es vorkommen, dass Bedingungen erfüllt sein müssen, bevor eine Instruktion ausgeführt werden darf. Es muss z. B. sichergestellt sein, dass eine Einverständniserklärung vorliegt, bevor der Patient operiert wird.

3. Abbrechen von Aktionen

Es kann unter bestimmten Bedingungen vorkommen, dass Instruktionen nicht ausgeführt werden dürfen. Eine mögliche Situation wäre z. B., dass ein zur Operation angemeldeter Patient berichtet, sich kurz vor der Aufnahme an einem fibrigen Infekt infiziert zu haben, was dazu führt, dass eine Narkose unmöglich wird.

Formalisierung $P : R \times \{true, false, n.A.\} \rightarrow \{SKIP, WAIT, ABORT, IGNORE\}$,
mit $R = \text{Basisregeln}$

- P, Menge der definierbaren Ablaufregeln
- R, Menge der Basisregeln

Beschreibung Ablaufregeln erweitern die Basisregeln, indem sie der Auswertung der Basisregel, eine Folge zuordnen. Eine Ablaufregel enthält die Information darüber, welche Basisregel mit welchem Ergebnis zu welcher Folge für den Workflow ausgewertet wird. Analog zu Basisregeln sind Ablaufregeln eine Art Baustein für den fachlichen Entwickler, die eine oder mehr Basisregeln als Parameter erwarten und denen, bei zutreffendem Ergebnis (wahr) eine der Folgen „SKIP“, „ABORT“, oder „WAIT“ zugeordnet werden, wobei das zusätzliche Ergebnis „ignore“ dem „WAIT“ entspricht. Der Wert dient dazu, das Ergebnis „nicht-anwendbar“ aus Basisregeln aufzufangen, sofern dies gewünscht ist. Dieses Hilfskonstrukt wird verwendet, falls es notwendig wird, dass ABORT-Regeln explizit geprüft werden müssen, z. B. falls Fälle i.S.v. „Es muss sichergestellt werden, dass der Patient nicht ...“ eintreten. Diese müssen von Fällen, wie z. B. „Bei Bekannter Allergie gegen ...“ abgrenzbar sein. Deshalb erhalten ABORT- und SKIP-Regeln einen zusätzlichen Markierer „notwendig“, der dem technischen Entwickler als Option zur Verfügung gestellt wird. Dieser führt dazu, dass SKIP- und ABORT-Regeln, falls Werte fehlen, wie nicht erfüllte WAIT-Regeln behandelt werden. Somit wird die Ausführung verzögert, bis alle Werte vorhanden und alle Vorbedingungen erfüllt sind.

3.3.4.3 Überwachungsregeln

Motivation Die Überwachungsregeln dienen dazu, sicherzustellen, dass bei der Ausführung von Workflowelementen bestimmte Rahmenbedingungen überwacht werden. Gerade im medizinischen Umfeld lassen sich oft nicht alle Details vorhersehen bzw. im Voraus modellieren oder es treten Komplikationen ein, die durch einen Mediziner persönlich gelöst

weren müssen. Abgesehen davon kann es vorkommen, dass bei der automatischen Ausführung eines Workflows nicht feststeht, wann welche Schritte erreicht sein müssen. Dieses Problem wird durch die Überwachungsregeln gelöst, da sie die Ausführung des Workflows beobachten und bei bestimmten Problemen, die sich anhand bestimmter Regeln identifizieren lassen, entsprechende, vordefinierte Aktionen auslösen. Eine Regel könnte z. B. die Überwachung von erwarteten Nebenwirkungen sein. Nach einer Transplantation kann eine erhöhte Anzahl an Antikörpern auf eine Abstossungsreaktion hinweisen. Überwachungsregeln erlauben es hier, für entsprechende Messwerte Grenzen festzulegen, ab denen eine Warnung oder eine andere Instruktion ausgelöst wird. So kann z. B. festgelegt werden, dass schon einmal eine Nachricht gesendet wird, ein Operationsaal für eine Notoperation angefordert wird und die Station informiert wird, dass ggf. Angehörige informiert werden müssen.

Beschreibung Überwachungsregeln sind an ECA-Regeln (Event-Condition-Action) angelehnte Konstrukte, wobei allerdings der Event auf das Prüfen der Regel beschränkt ist. Diese Regeln dienen dazu, Fehler vor, nach oder während der Ausführung einer Instruktion zu erfassen und entsprechende Maßnahmen einzuleiten.

Ein mögliches Einsatzgebiet ist das Einhalten von Zeitfenstern, d. h. es könnte eine WAIT-Regel formuliert worden sein, die dafür sorgt, dass erst mit Aktionen begonnen wird, sobald bestimmte Laborergebnisse vorliegen. Sollten diese aufgrund unvorhersehbarer Ereignisse nie eintreffen, würde der Workflow nicht fortgesetzt. Hier würde die Formulierung einer Überwachungsregel in der Form „Wenn nach 150 Minuten keine Ergebnisse vorliegen -> Behandelnden Arzt benachrichtigen“ verhindern, dass der Workflow zum völligen Stillstand kommt.

Umgesetzt werden die Überwachungsregeln durch bereits bestehende Konstrukte, konkret aus Regeln und Instruktionen. Dabei werden Regeln in einem Baustein mit Instruktionen kombiniert, die ausgelöst werden, wenn die Regeln zutreffen. Eine Überwachungsregel unterscheidet sich von ECA-Regeln dadurch, dass sie nicht durch das Eintreten eines bestimmten Events ausgewertet werden, sondern an einer bestimmten Stelle im Ablauf des Workflows.

3.3.5 Aktionspläne

Motivation

Mit den Ablaufregeln, Instruktionen und Überwachungsregeln ließen sich Workflows vollständig konstruieren. Aufgrund der Tatsache, dass dieser Ansatz ad hoc modifizierbare Workflows unterstützt, muss der Aufbau des Workflows möglichst einfach gestaltet werden können. Dies umfasst eine einfache „Gruppierung von Elementen“. Es muss somit erkenntlich sein, welche Elemente logisch zu einer größeren Aktion (im Sinne einer Zusammenfassung kleinerer Handlungen oder Aufgaben) zusammen gehören, d. h. es muss zu identifizieren

sein, welche Elemente die gleichen Vorbedingungen haben.

Da die Modellierung eines gesamten Workflows, allein aus Instruktionen und Regeln zu feingranular wäre, stellt der Aktionsplan ein Mittel dar, komplexe Abläufe gebündelt aufzubauen.

Ein Anwendungsbeispiel ist die Aufnahme eines Patienten, bestehend aus dem Erfassen der nicht vorhandenen Daten, der Weiterleitung zur Anamnese, ggf. der stationären Aufnahme, dem Senden einer Nachricht an die Stationsbereitschaft über die Aufnahme eines Patienten und dem Ausdruck von Barcodes für Patientenkontrollblätter, sowie Laufzetteln und Zusatzinformationen. Jede dieser Aktionen stellt eine Instruktion dar. Alle zusammen können in einem Aktionsplan "Aufnahme" gebündelt werden, welcher im Folgenden als ein Element im Workflow verwendet wird. Der Vorteil hierbei ist, dass ein solcher Aktionsplan als Container in verschiedenen Situationen wiederverwendet werden kann.

Formalisierung

$A := (P_a, I_a, O_a)$, mit $P_a \subseteq P, I_a \subseteq I, O_a \subseteq O$

P: Ablaufregeln

I: Menge der Instruktionen

O: Überwachungsregeln

Beschreibung

Aktionspläne fassen eine Instruktionsgruppe mit Ablaufregeln als Vorbedingungen und Überwachungsregeln in einem Container zusammen. Dies hat folgende Vorteile: Zum einen ist es somit leichter ersichtlich, aus welchen kleineren Arbeitsschritten sich ein komplexerer Arbeitsschritt zusammensetzt. Zum anderen lässt sich ein so konstruiertes Element leichter wieder verwenden, da es die Vorbedingungen seiner Ausführung und die Instruktionen, die damit verbunden sind bündelt.

Ein Workflow entsteht jedoch erst, durch die Kombination mehrerer Aktionspläne. Die Instruktionen in einem Aktionsplan sind durch die Verwendung einer Instruktionsgruppe lediglich in einer Halbordnung angeordnet, eine komplexere Anordnung der Elemente ist nicht vorgesehen. Die Hauptaufgabe eines Aktionsplans besteht darin, einen Arbeitsschritt, zu beschreiben und dabei gleichzeitig die notwendigen Vorbedingungen zu prüfen (durch Ablaufregeln), sowie die Ausführung zu überwachen (durch Überwachungsregeln).

3.3.6 Automatische Entscheider

Motivation

Aktionspläne enthalten die Bedingungen für die Ausführung von Instruktionen bereits in den Ablaufregeln. Bei der Modellierung von klinischen Pfaden und medizinischen Leitlinien,

gelten Bedingungen nicht nur für bestimmte Behandlungsschritte (Aktionspläne), sondern insbesondere auch für Pfade, die z. B. Handlungsalternativen beschreiben. Solche Vorbedingungen könnten in den Aktionsplänen zu Beginn eines Teilpfades abgebildet werden, dies hätte allerdings zwei gravierende Nachteile. Zum einen wäre die Wiederverwendbarkeit des Elementes in dem die Bedingung für den Pfad eingearbeitet wurde nicht mehr gegeben, da ein Einsatz dieses Elementes in einem anderen Pfad eine Modifikation bedeuten würde. Zum anderen wird nach außen nicht dargestellt, dass es sich um Teilpfade handelt, für die besondere Bedingungen gelten. Von daher bietet sich die Definition eines Elementes an, das explizit Bedingungen für Teilpfade repräsentiert, und somit automatisch Entscheidungen für die Aktivierung oder die Deaktivierung eines Teilpfades trifft.

Beschreibung

Bei automatischen Entscheidern handelt es sich um Aktionspläne, die nur aus Ablaufregeln und Überwachungsregeln bestehen. Dies bietet wiederum den Vorteil, dass sie aus Standardkomponenten bestehen. Abgesehen davon können zum einen bestimmte Zustände des Patienten anhand von Ablaufregeln erkannt werden, zum anderen kann durch entsprechende Überwachungsregeln darauf reagiert werden.

Für dieses Element stehen nur ABORT- und WAIT-Regeln zur Verfügung, da es keine Instruktionen enthält, die übersprungen werden könnten. Eine SKIP-Regel würde alle WAIT-Regeln ausser Kraft setzen und das Element überspringen. Da der Einsatz von SKIP-Regeln sehr fehleranfällig ist, wird an dieser Stelle die Einschränkung vorgenommen, auf SKIP-Regeln zu verzichten.

Beim Einsatz in einem Workflow, stellt ein automatischer Entscheider sicher, dass der Pfad, der auf ihn folgt, die Bedingungen erfüllt, die in seinen Ablaufbedingungen formuliert wurden. Es besteht auch die Möglichkeit, automatische Entscheider an die Spitze mehrerer konkurrierender Teilpfade zu stellen und somit eine automatische Pfadentscheidung innerhalb des Workflows zu modellieren. Allerdings muss hierbei sichergestellt sein, dass die Regeln konsistent formuliert sind. Sollten sich bestimmte Pfade gegenseitig ausschließen, müssen die Regeln entsprechend (disjunkt) formuliert sein.

3.3.7 Manueller Entscheider

Motivation

Nicht immer ist es gewünscht, dass Pfadentscheidungen automatisch durch den modellierten Workflow getroffen werden. Manchmal ist es notwendig oder auch einfacher, wenn der Anwender direkt über einen bereits modellierten Pfad und dessen Ausführung entscheiden kann. Demnach wird ein weiteres Element benötigt, das die Entscheidung für die Abarbeitung eines Teilpfades an den Anwender delegiert und über eine Schnittstelle die Möglichkeit bietet, zwischen verschiedenen Pfaden zu wählen.

Beschreibung

Der manuelle Entscheider ist ein Element, das aus mehreren benannten Ausgängen besteht, an denen die verschiedenen Teilpfade unter denen eine Entscheidung getroffen werden kann, angehängt sind. Je nach Auswahl des Anwenders werden die entsprechenden Teilpfade abgearbeitet.

Eine Alternative Umsetzung dieses Entscheiders wäre ein Aktionsplan mit einer Instruktion i.S.v. „setzeVariable(Pfadentscheidung)“ und den nachfolgenden Teilpfaden jeweils einen automatischen Entscheider voranzusetzen. Diese würden die Variable „Pfadentscheidung“ auswerten und den entsprechenden Pfad aktivieren. Es wurde jedoch entschieden, den manuellen Entscheider als ein neues Element einzuführen. Zum einen, um die Modellierung einfacher und übersichtlicher zu gestalten und zum anderen, um manuelle und automatische Entscheidungen besser voneinander trennen zu können.

3.3.8 Workflow

Motivation

Der Workflow bildet den zu modellierenden klinischen Pfad ab, den ein Patient mit einer bestimmten Symptomatik durchläuft.

Formalisierung

Formal ist ein Workflow in diesem Konzept ein kreisfreier gerichteter Graph $W := (K, B, S_0, S_e)$ mit $B \subseteq BasisElemente = \text{Aktionspläne} \cup \text{Automatische Entscheider} \cup \text{Manuelle Entscheider} \cup \text{Synchronisationspunkte}$ und $K = \text{Menge an gerichteten Kanten von } i \text{ nach } j: (i, j) \text{ mit } i \in B \cup \{S_0\} \wedge j \in B \cup \{S_e\}$
 $S_0 := \text{Startpunkt}$
 $S_e := \text{Endpunkt}$

Beschreibung

Der Workflow ist somit ein Konstrukt, das durch die Kombination seiner Elemente zu einem Graphen wird, der einen klinischen Pfad bzw. eine medizinische Leitlinie abbildet. Dazu werden die Elemente aus denen der Graph besteht (Aktionspläne, Entscheider, Synchronisationspunkte) mit gerichteten Kanten verbunden. Die Ausführungssemantik der Elemente wird in 3.3.9 genauer beschrieben.

3.3.8.1 Synchronisationspunkte

Motivation

Um die Ausführung von Teilpfaden, unabhängig von den geschriebenen oder gelesenen Daten synchronisieren zu können, wurden die Synchronisationspunkte eingeführt. Obwohl Aktionspläne synchronisiert werden könnten, indem die gleiche WAIT-Regel in verschiedenen Aktionsplänen eingesetzt wird, ist es unter Umständen erforderlich, die Ausführung von Aktionsplänen direkt von der Beendigung vorhergegangener Pfade abhängig zu machen.

Beschreibung

Alle mit dem Synchronisationspunkt verbundenen Kanten müssen aktiviert sein bevor die nachfolgenden Elemente aktiviert werden. Somit entspricht ein Synchronisationspunkt einem And-Join.

3.3.8.2 Aufbauregeln

Motivation Ein besonderes Problem bei der Ad-hoc-Modellierung ist, dass ein Workflow nicht „aus einem Guss“ gefertigt wird. Der Modellierer muss zum Teil auf Bausteine zurückgreifen, die jemand anderes definiert bzw. zusammengestellt hat. Das eigentliche Problem besteht hier in dem fehlenden Wissenstransfer. Ein Fachanwender kennt z. B. Abhängigkeiten zwischen Behandlungen und Behandlungsschritten. Beispielhaft zu nennen wären hier notwendige Voruntersuchungen, bei bestimmten Behandlungen. Bei der Kombination von Elementen zu Workflows, gibt es jedoch auch aus technologischer Sicht fachliche Bedingungen, die dem Aufbau zu Grunde liegen, z. B. das Erfassen der Patientendaten, vor dem Schreiben eines Arztbriefes. Es werden also Elemente benötigt, die den Fachanwender bei der sinnvollen Kombination von Elementen unterstützen.

Beschreibung Im Kapitel 3.2.5 wurde ein Ansatz vorgestellt die Kombination von Elementen im Workflow mit Hilfe von Kontrollfluss-Bedingungen zu beschreiben. Dieser musste allerdings für dieses Konzept modifiziert werden. Anders als bei den Bedingungen von denen im o.g. Abschnitt ausgegangen wird, werden hier vorher nicht bekannte Elemente, zu diversen, im Voraus nicht bekannten Workflows zusammengesetzt. Mit Hilfe der Kontrollfluss-Bedingungen, lässt sich die Gültigkeit von Workflows, im Sinne der dafür definierten Regeln formal prüfen. Der Ansatz, eine Menge von Regeln zu definieren und diese auf Korrektheit zu prüfen, würde bei einer unbekanntem Anzahl von Elementen und Regeln zu Problemen führen. Ein besonderes Problem ergäbe sich bei der Optimierung und Minimierung der Regelmengen, da diese zum Teil auf NP-vollständigen Problemen basieren. Hierbei kann nicht garantiert werden, dass die Anzahl der Regeln klein genug bleibt, um

effizient bearbeitet werden zu können. Für dieses Konzept ist es wichtig dass die Regeln an die Elemente gebunden sind.

Das heisst, dass Aufbauregeln nur dann zum Tragen kommen, wenn entsprechende Elemente verwendet werden. Eine Regel, die beschreibt, dass Aktionsplan A vor Aktionsplan B erfolgen muss, wird nur ausgewertet, wenn Aktionsplan B eingesetzt wird. Allerdings muss in diesem Konzept zwischen zwei verschiedenen „Adressaten“ für Aufbauregeln unterschieden werden, die sich in ihrer Komplexität unterscheiden:

Für Workflows: Regeln werden bei der Kombination der Elemente, d. h. beim Aufbau des Workflows verwendet. Die Regeln werden durch einen technischen Entwickler oder Fachanwender, an die Elemente annotiert. Damit stehen jeweils zwei Parameter für die Regeln bereits zur Modellierungszeit fest. Der Workflow in dem die Regel angewendet wird und das Element auf das sich eine Regel bezieht. Somit kann bei der Adaption der in Abschnitt 3.2.5 beschriebenen Kontrollfluss-Bedingungen, auf diese Parameter verzichtet werden.

- Include
Include(E) Diese Regel besagt, dass ein Element E Teil des Workflows sein muss.
- Exclude
Exclude(E) Diese Regel besagt, dass ein Element E nicht Teil des Workflows sein darf.
- Fork
Fork(E₁, ..., E_n)
Diese Regel besagt, dass mehrere Elemente $E_1 \dots E_n$ parallel dem Element folgen, dem diese Regel zugeordnet wird. Im Falle von Entscheidern, sollte hier noch die Pfadbezeichnung des Pfades annotiert sein, auf dem das Element folgt.
- Serial
Die Regel *Serial(E₁, ..., E_n)* legt fest, dass zwar alle Elemente $E_1 \dots E_n$ ausgeführt werden müssen, die Reihenfolge aber nicht festgelegt ist. (Dabei ist das aktuelle Element zwar nicht in E_1, \dots, E_n enthalten, wird aber bei der Auswertung hinzugefügt.)
- Order
Order(E, position), mit position $\in \{vorher, nachher\}$
Diese Regel legt fest das E vor, bzw. nach dem betrachteten Element ausgeführt werden soll. Durch die Annotation an ein Element, ist hier die Reihenfolge wichtig. Allerdings ist es leicht einzusehen, dass die Regeln sich zur Modellierungszeit einfach auf die in Abschnitt 3.2.5 beschriebenen „Order-Bedingungen“ abbilden lassen.

Mit einem solchen mächtigen Werkzeug ließen sich sogar medizinische Leitlinien modellieren, indem die Aufbauregeln dazu benutzt werden, den Aufbau eines Behandlungspfades zu definieren. Bei der Beschränkung auf Dummy-Elemente, die auf reale Elemente abgebildet werden können, ließe sich sogar die Leitlinienkonformität eines Workflows verifizieren.

Für Aktionspläne: Aktionspläne sind das zweite Element, das aus kleineren Unterelementen besteht. Hierbei ist jedoch die Möglichkeit der Anordnung der Elemente begrenzter, als bei Workflows, weil bei Aktionsplänen die Instruktionen nur gemäß ihrer definierten Halbbordnung ausgeführt werden. Daraus folgt, dass auf Serial- und Fork-Regeln verzichtet werden kann. Somit reduziert sich die Anzahl der Regeln auf:

- Include
Include(E) Diese Regel besagt, dass ein Element E Teil des Aktionsplans sein muss. Besonders hierbei ist, dass sich ein Element auch auf Ablaufregeln beziehen kann, womit ermöglicht wird, dass Instruktionen durch bestimmte Regeln abgesichert werden.
- Exclude
Exclude(E) Diese Regel besagt, dass ein Element E nicht Teil des Aktionsplans sein darf. Es gilt ebenfalls, dass sich ein Element auch auf Ablaufregeln beziehen kann, womit ermöglicht wird dass Instruktionen durch bestimmte Regeln abgesichert werden.
- Order
Order(E, position), mitposition ∈ {vorher, nachher}
Diese Regel legt fest, dass ein Element E, wie für Aktionspläne auch, vor, bzw. nach dem betrachteten Element ausgeführt werden soll, wobei das Element E hier auf Instruktionen und Instruktionsgruppen begrenzt ist.

Anwendung von Regeln

Order- und Serial-Regeln werden nur ausgewertet, wenn auch die Elemente, auf die sie sich beziehen, im Workflow vorhanden sind. Erst eine Kombination dieser Regeln mit Include() sollte zu einer Fehlermeldung führen. Regeln sollen primär der Unterstützung des Benutzers, im Hinblick auf das, beim Erstellen der Elemente, notwendige Fachwissen (technisch oder medizinisch) dienen. Von daher bietet es sich an, an die Regeln Metainformationen zu annotieren, die beschreiben, welche Bedeutung die jeweilige Regeln hat. Eine stricke Anforderung der Einhaltung der Regel steht jedoch im Widerspruch zu den Grundsätzen dieses Konzeptes. Da prinzipiell davon ausgegangen werden muss, dass der Fachanwender über mehr Wissen verfügt, als das System, sollte das System den Anwender nicht einschränken, sondern lediglich auf die Bedeutung der Vorschläge hinweisen, damit er die Anwendung der Regeln selbst beurteilen kann. Ein Fachanwender könnte z. B. Kenntnis über Behandlungen haben, die nicht im Workflow erfasst sind, jedoch trotzdem Daten produziert haben.

Weiterer Vorteil bei der Modellierung

Durch die Include- und Exclude-Regeln lassen sich dem Benutzer Vorschläge für weiter zu verwendende Elemente machen und diese in einer Übersicht bereitstellen. In Anlehnung an

die aus dem Online-Handel bekannten Kaufvorschläge, könnte ein Hinweis in etwa wie folgt dargestellt werden: „Wenn Sie dieses Element benutzt haben, sollten Sie auch die folgenden Elemente verwenden.“

3.3.8.3 Varianten

Varianten sind Folgen vordefinierter Ad-hoc-Operationen auf Workflows. Varianten können als kleine Scripte zum Umbau von Workflows verstanden werden, die vom fachlichen Entwickler, bei der Modellierung des Workflows schon vorgesehen werden können und durch die Anwender ausgeführt werden. Ein Anwender kann somit Anpassungen am Workflow vornehmen, ohne in die Rolle des fachlichen Entwicklers schlüpfen zu müssen. Dazu muss vor der Anwendung einer Variante vom System geprüft werden, ob alle, in der Variante anzuwendenden Operationen, gültig sind. Ist dies der Fall, kann der Anwender die Variante auswählen. Der Vorteil von Varianten liegt vor allem darin, dass der Workflow nicht zu komplex wird und häufige Behandlungsalternativen nicht in verschiedenen Workflows bereit gestellt werden müssen. Ein weiterer Vorteil ist, dass Varianten auch noch bis zu einem gewissen Punkt des Workflows anwendbar sind. Beispielhaft für eine Behandlung, für die sich die Modellierung in Varianten anbietet, ist die Ausführung von Zusatzuntersuchungen, die bei seltenen Indikationen notwendig werden. Ferner wäre es auch denkbar, Varianten bei auftretenden Schwierigkeiten vor einer Operation anzuordnen. Auch Optionen, die durch individuelle Gesundheitsleistungen angeboten werden können, lassen sich durch Varianten abdecken.

3.3.9 Ausführungssemantik der Elemente

Workflow

In einem Workflow werden die Elemente sukzessive von ihrem Vorgänger aktiviert, folgende Knoten bilden Ausnahmen:

1. Startknoten
Bei der Instanzierung eines Workflows werden automatisch alle mit diesem Punkt, durch ausgehende Kanten, verbundene Elemente aktiviert.
2. Synchronisationspunkte
Ein Synchronisationspunkt hat Vorgänger und Nachfolger. Hier erfolgt die (synchrone) Aktivierung aller Nachfolger erst, wenn alle Vorgänger beendet wurden.
3. Endpunkt
Der Endpunkt dient als Marke, die signalisiert, dass der Workflow abgeschlossen ist. Dies bedeutet nicht, dass alle noch nicht abgeschlossenen Elemente des Workflows abgebrochen werden. Vielmehr ist dieses Konstrukt bereits jetzt in das Design eingegangen, weil es im Hinblick auf verschachtelte Workflows sinnvoll sein könnte, einen

nachfolgenden Workflow schon einzuleiten, obwohl noch nicht alle, außer den wichtigen Workflowprozessen abgeschlossen sind. Beispielhaft wäre hier das Aufräumen nach einer Operation, das sicherlich in eine Modellierung hinein gehört, aber auf die Workflows, in denen sich der aktuelle Patient befindet, keine Auswirkungen hat. Man kann also in dem Workflow „Operationssal putzen“ den Prozess schon nach dem Putzen an sich als beendet ansehen, danach könnte z. B. noch das Schreiben eines Berichts folgen. Setzt man also den Endpunkt zwischen Putzen und Bericht schreiben, signalisiert man damit, dass der Workflow in den wichtigen Punkten schon abgeschlossen ist, und nicht auf andere Workflows zurückwirkt. Der erstellte Bericht würde dann automatisch an den benötigten Stellen (z. B. Auswertung aller Berichte) zu Synchronisationseffekten führen.

Als wirklich abgeschlossen gilt ein Workflow dann, wenn keine Elemente mehr aktiviert sind. Bei der Modellierung von Workflows muss für Elemente deren Abschluss trotz Aktivierung sichergestellt werden. Eine ABORT-Regel „Workflow-Ende erreicht?“ eingeführt werden. Diese prüft, ob alle Elemente, die abgeschlossen sein müssen und diese Regel nicht enthalten, abgeschlossen sind. Um die Modellierung für den Anwender einfacher zu gestalten, bietet es sich an, die Existenz dieser Regel durch eine Checkbox, im Element selbst, bereit zu stellen.

Der Vorteil einer solchen Ausführungssemantik liegt in der erreichbaren Flexibilität und Übersichtlichkeit. Somit ist sichergestellt, dass auch komplexe Prozesse abgebildet werden können, die gerade im Klinikalltag von verschiedenen Personen durchgeführt werden und von vielen Entscheidungen abhängen, die oft erst während einer Behandlung getroffen werden können. Man denke z. B. an die Stellung von Diagnosen oder im Behandlungsablauf auftretende Komplikationen. Hier können verschiedene Alternativen und optionale Untersuchungen entsprechend verschachtelt und je nach Entscheidung eingebunden oder übersprungen werden. So lassen sich sowohl sequentielle als auch parallele Abläufe modellieren und es ist sichergestellt, dass sich Teilschritte nicht überholen, z. B. Übergabe eines Patienten, bevor Untersuchungsergebnisse vorliegen.

Aktionsplan

Sobald ein Aktionsplan aktiviert wird, beginnt die Überprüfung der Regeln. Hierbei werden zunächst die Ablaufregeln betrachtet, deren Auswertung der Basisregel zu folgenden Ergebnissen führen kann:

1. ABORT

Diese Bedingung führt dazu, dass der gesamte Teilpfad ab diesem Aktionsplan beendet wird.

Beispielregel: Patient nicht anästhesiefähig → OPVorbereitung abbrechen.

2. SKIP

Wurde eine Regel als SKIP-Bedingung formuliert, wird der aktuelle Aktionsplan bei

Zutreffen der Regel übersprungen. Dies bedeutet, dass die enthaltenen Instruktionen nicht ausgeführt werden, der Aktionsplan wird aber als beendet markiert und die Nachfolger entsprechend aktiviert .

Beispielregel: Kardiologische Vorerkrankung → EKG erstellen („wenn nicht, dann nicht“).

3. WAIT

Hierbei wird mit der Ausführung der Instruktionen so lange gewartet, bis die Bedingung positiv erfüllt wurde.

Beispielregel: Röntgenbild liegt vor und ist ohne Befund → Behandlung beginnen.

4. IGNORE

Diese Regel muss erneut geprüft werden, da das Ergebnis nicht Verwertbar ist, weil z.B. ein Wert nicht vorhanden ist.

Die Überprüfung der Ablaufbedingungen erfolgt nach der Aktivierung des Aktionsplans und in der obigen Reihenfolge. Die Auswertungsreihenfolge von Aktionsplänen begründet sich in deren Priorität für den Patienten. Die Motivation der Regeln liegt vor allem in der Abbildung von Fachwissen. Hier geht ein Schutz des Patienten vor, weshalb Abbruchbedingungen, die z. B. Kontraindikationen für eine Behandlung modellieren zuerst ausgewertet werden. Beim Zutreffen einer Abbruchbedingung wird der Workflow an dieser Stelle beendet, und nachfolgende Elemente werden nicht ausgewertet. Als nächstes werden die SKIP-Regeln ausgewertet. Wird eine Regel erfolgreich nach „SKIP“ ausgewertet, wird der aktuelle Aktionsplan übersprungen. Dieses Verhalten hat zwei Vorteile, zum einen lassen sich dadurch nur bedingt auszuführende Aktionspläne einfach modellieren, so kann z. B. ein Aktionsplan für einen Bluttest fest eingefügt und in dessen Regel festgelegt werden, dass der Aktionsplan nur dann ausgeführt wird, wenn kein aktueller Blutwert vorliegt. Zum anderen ersparen SKIP-Regeln unnötige Arbeiten, z. B. Arbeiten die in anderen Workflows, oder am Workflow vorbei schon ausgeführt wurden und sich dies an einer Variablen (z. B. einem Bluttestergebnis) nachvollziehen lässt. Als letztes werden die WAIT-Regeln ausgewertet. Sie dienen dazu bestimmte Bedingungen vor dem Beginn der Abarbeitung der Instruktionen sicher zu Stellen.

Sind alle Regeln ausgewertet, trafen keine ABORT- und keine SKIP-Regeln zu und sind alle WAIT-Regeln zu wahr ausgewertet worden, beginnt die Abarbeitung der Instruktionen. Parallel zu der Prüfung der Ablaufregeln und der Abarbeitung der Instruktionen werden die Überwachungsregeln immer wieder ausgeführt. Da sie die Aufgabe haben sicher zu stellen, dass die in den Regeln definierten Bedingungen während der Ausführung des Aktionsplans gelten, müssen sie parallel in einer Schleife ausgeführt werden bis der Aktionsplan abgeschlossen ist. Trifft eine Überwachungsregel zu „deaktiviert“ sie sich selber und führt die ihr zugewiesene Instruktion aus.

Automatische Entscheider

Bei einem automatischen Entscheider handelt es sich im Großen und Ganzen um einen Aktionsplan, der keine Instruktionen enthält. Für die Ausführungssemantik des Elements bedeutet das, dass nach der Auswertung der Regeln das Verhalten dem eines Aktionsplans entspricht. Allerdings werden nach der Erfüllung der WAIT-Regeln alle nachfolgenden Elemente aktiviert.

Manuelle Entscheider

Dieses Element delegiert die Entscheidung über die Aktivierung nachfolgender Pfade an den Benutzer. Nach der Rückmeldung, der zu aktivierenden Pfade durch den Benutzer, werden diese Pfade (ausgehende Kanten) aktiviert.

Ad hoc Operationen

Bei der Entwicklung dieses Konzeptes wurde besonderer Wert auf die Realisierung von ad hoc Änderungen auf Workflows gelegt. Bei Änderungen an Workflows muss zwischen Änderungen auf Modellebene und Änderungen auf Instanzebene unterschieden werden. Unter der Modellebene wird die Vorlage auf der der Workflow beruht verstanden. Die Instanz hingegen umfasst, neben dem zugrunde liegenden Modell, auch die aktuelle Konfiguration, also die Information, in welchem Zustand sich der Workflow gerade befindet. Jedes Schreiben, bzw. Lesen eines Datums, sowie jedes Fortschreiten im Workflow ist demnach eine Änderung der Konfiguration.

Eine Änderung an einer Workflow-Instanz führt immer zu einer Änderung des zugrunde liegenden Modells. Eine der Herausforderungen bei den Änderungsoperationen ist deshalb die Konsistenz von Workflow-Instanz und Modell zu sichern. Um dies zu gewährleisten wird die Ausführung, aller aktiven Instruktionen unterbrochen, somit werden keine Daten gelesen und geschrieben, allerdings werden eingehende Werte zwischengespeichert. Somit wird sichergestellt, dass, falls der Workflow während einer Abfrage unterbrochen wird, keine Daten verloren gehen. Eine Abfrage kann sowohl eine Datenbanktransaktion als z. B. auch das Warten auf eine E-Mail oder ähnliches darstellen. Wird jegliche Ausführung des Workflows unterbrochen, lässt sich der Workflow, bis auf die Elemente, die bereits ausgeführt wurden, wieder in Modell und Konfiguration trennen.

Werden jetzt Änderungen am Modell vorgenommen, müssen diese mit der bisherigen Konfigurationsfolge konform sein. Dies hat zwei Gründe:

1. Gerade in einem so kritischen Umfeld, wie einem Krankenhaus, muss die Dokumentation der Historie sichergestellt sein.
2. Die Manipulation von bereits angefangenen Instruktionen oder Änderungen an bereits abgearbeiteten Ablaufregeln können zu Inkonsistenzen zwischen Modell und Instanz führen, die jede Nachvollziehbarkeit des Workflowablaufs behindern.

Eine Anforderung an Ad-hoc-Modifikationen muss daher sein, Änderungen in bereits abgeschlossenen Elementen und in allen Instruktionen die gerade abgearbeitet werden zu verbieten. Dies bezieht sich auch auf Ablaufregeln eines Workflows, wenn diese bereits ausgewertet wurden und zur Ausführung der Instruktionen bzw. zum Abbrechen oder Überspringen des Aktionsplans geführt haben.

Wurde die Konsistenz des Workflows wie oben beschrieben sichergestellt, kann der noch nicht abgearbeitete Teil als eigenständiges Workflow-Modell mit den bereits aktiven Aktionsplänen als Startpunkte, aufgefasst werden. Dieses Modell läßt sich vom fachlichen Entwickler anpassen, wie jedes andere Modell. Dabei stehen folgende Operationen zur Verfügung:

- Einfügen von Elementen
In den Workflow lassen sich Aktionspläne, Entscheider, Kanten und Synchronisationspunkte einfügen. Sichergestellt werden muss dabei, dass keine ausgehende Kanten von einem Modell zu einem bereits abgearbeiteten Element führt.
Bei Einfügeoperationen in Aktionsplänen dürfen neue Instruktionen in der Ordnung nicht vor bereits abgearbeiteten Elementen oder gerade ausgeführten Elementen eingefügt werden.
- Löschen von Elementen
Generell können alle Elemente aus dem noch nicht abgearbeiteten oder gerade ausgeführten Teil des Workflows entfernt werden. Beim Löschen eines Elementes werden quasi als Vorschlagwert, alle vorhergehenden Elemente mit allen, durch die Ausgangskanten verbundenen Elemente verbunden, was der natürlichen Abfolge entspricht. Somit wird das Element dadurch, aus allen diesen Pfaden entfernt. Ausnahme bilden hier manuelle Entscheider, da hier der Pfaddurchlauf an den ausgehenden Kanten von dem Element selbst bestimmt wird und somit die Pfade, die durch das Element verlaufen nicht identifizierbar sind.
- Verschieben von Elementen
Das Verschieben von Elementen lässt sich auf das Löschen und Einfügen zurückführen.
- Editieren von Elementen
Das Editieren von Elementen bezieht sich auf Regeln und Instruktionen, die parametrisierbar sind. Hier lassen sich die Parameter, wie z. B. Daten, die bearbeitet werden ändern, so lange das Element noch nicht bearbeitet wurde.
- Abbrechen von Elementen
Eine weitere Möglichkeit des Eingreifens wäre das Abbrechen von Aktionsplänen. Da Aktionspläne und die darin enthaltenen Instruktionen die einzigen Elemente des Workflows sind, in denen Anweisungen ausgeführt werden, bezieht sich das Abbrechen eines Aktionsplan immer auf die noch nicht ausgeführten Instruktionen. Das

Abbrechen einer Instruktion entspricht dem Entfernen aller noch nicht ausgeführten Instruktionen und dem Abschliessen des Aktionsplans.

3.3.10 Korrektheitskriterien

Da aus diversen Gründen davon abgesehen wurde, den Datenfluss explizit zu modellieren (vgl. Kapitel 3.3.1.2), wird von der Betrachtung der Korrektheit des Datenflusses ebenfalls abgesehen. Es wird lediglich davon ausgegangen, dass der Datenfluss dem Kontrollfluss zu folgen hat (vgl. [102, S. 268f]). Hiermit wird sichergestellt, dass fehlende oder falsch erhobene Daten, den Ablauf des Workflows nicht behindern. Mit den Aufbauregeln (vgl. Kapitel 3.3.8.2) wurde dem Entwickler jedoch eine Möglichkeit gegeben, Regeln zu definieren, die genau solche Bedingungen bei Bedarf an den Workflow stellen. Trotzdem sei hier noch einmal darauf hingewiesen dass entschieden wurde, den Datenfluss in den Bausteinen zwar anzuzeigen, die Korrektheit jedoch nicht zu erzwingen. Hiermit soll den Modellierern die Möglichkeit gegeben werden, Dateneingaben in die „Datenwolke“ aus anderen Quellen zu antizipieren und den auf die Datenmodellierung bewusst mit aufzunehmen. Es soll also der Freiraum für Modellierungslücken offen bleiben. Es wäre nicht notwendig Daten, die z. B. in einem Schichtprotokoll bei jedem Schichtwechsel automatisch protokolliert werden in jeden Workflow mit einzumodellieren. Abgesehen davon, würde die parallele Ausführung mehrerer Workflows für einen Patienten auf den gleichen Daten behindert werden. Ein Patient hat per Definition zu bestimmten Zeitpunkten nur einen Datenzustand, z. B. die Körpertemperatur, die auch, wenn sie nicht in dem aktuellen Workflow erhoben wurde, auf diesen wirken kann. Denkbar wäre z. B., dass ein Medikament das in Workflow A verabreicht wurde dazu führt, dass ein Medikament in Workflow B nicht mehr verabreicht werden darf. Die Information, der Patient hat Medikament xy erhalten, wäre eine Information, die sich in alle Workflows vererben würde. Ein denkbare Szenario ist die so genannte Tromboosespritze, die in vielen liegenden Behandlungen zum Einsatz kommt, jedoch nicht doppelt verabreicht werden muss, falls sich ein Patient sich in zwei verschiedenen Workflows befindet (z. B. aufgrund von Multimorbidität).

Strukturelle Korrektheit (vgl. [102, S. 270ff])

Bei der Betrachtung von zunächst nur Instruktionen in Aktionsplänen, so ist, da sie (per Definition) eine Halbordnung bilden (vgl. Kapitel 3.3.5) klar, dass hier eine strukturelle Korrektheit gegeben ist. Wenn es eine gültige Halbordnung gibt, können die Instruktionen in dieser Reihenfolge ausgeführt werden. Nachdem ein Aktionsplan an sich natürlicher Weise strukturell korrekt ist, muss nun geprüft werden, ob dies auch in Workflows gegeben ist. Sei die Betrachtung zunächst nur auf Aktionspläne und Kanten dazwischen beschränkt, so kann jeder Aktionsplan mit seinen ausgehenden Kanten sowohl als AND-Split als auch OR-Join aufgefasst werden. Dies führt dazu, dass es mit diesen Grundkonstrukten unmöglich ist, strukturell nicht korrekte Workflows zu erstellen (Vgl. [102, S. 270ff]).

Wird die Betrachtung um Entscheider erweitert, so lassen sich auch XOR und „N aus M“ Konstrukte erzeugen. Indem mehrere, von einem Aktionsplan ausgehende Kanten zu Entscheidern weitergeleitet werden, lässt sich, je nach den gestellten Bedingungen ein XOR oder ein „N aus M“ Konstrukt erzeugen. Das gleiche Verhalten lässt sich mit einem manuellen Entscheider erzeugen, der eine „N aus M“ Auswahl ermöglicht. Aufgrund der Tatsache, dass bis jetzt nur OR-Joins möglich sind, ist sichergestellt, dass der Kontrollfluss korrekt ist. Hierbei wird vorausgesetzt, dass die Bedingungen in den Entscheidern korrekt sind. Um Fairnesskriterien zu erfüllen, muss gewährleistet sein, dass zumindest ein Entscheider „durchschaltet“. In diesem Fall ist immer noch klar, dass der Workflow korrekt ist, auch wenn die Korrektheit nun von den jeweiligen Bedingungen abhängt.

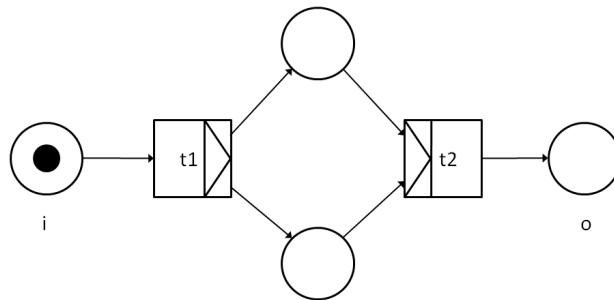


Abbildung 3.12: Beispiel eines Deadlocks in Petrinetzen. [102, S. 272]

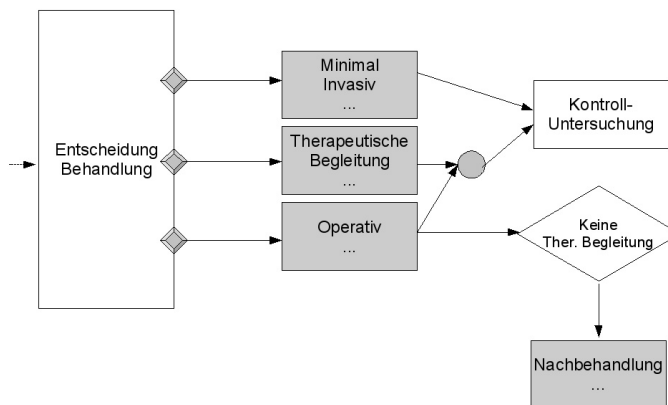


Abbildung 3.13: Beispielprozess

Ein Konstrukt, das in diesem Kontext dazu führen könnte, dass nicht korrekte Workflows erstellt werden können, sind Synchronisationspunkte. Der Grund hierfür ist, dass Synchron-

nisationspunkte AND-Joins entsprechen und somit ein Konstrukt, analog zu Deadlocks in Petrinetzen (siehe Abb. 3.12) erstellt werden kann. Ein Beispiel für eine solche Situation in diesem Konzept, ist in Abbildung 3.13 dargestellt. Die grau hinterlegten Aktionspläne stellen entweder ganze Pfade dar oder könnten durch solche ersetzt werden. Wird im manuellen Entscheider nur die „therapeutische Begleitung“ gewählt, so würde anschliessend keine Kontrolluntersuchung und auch keine Nachbehandlung durchgeführt. Der Workflow würde nach der „therapeutischen Begleitung“ nicht weiter ausgeführt.

In klassischen Workflows stellt diese Situation einen Deadlock dar, kann in medizinischen Workflows aber durchaus sinnvoll sein. Hierbei würde folgende Aussage modelliert: Immer wenn eine minimalinvasive Therapie durchgeführt oder ein operativer Eingriff therapeutisch begleitet wird, muss eine Nachuntersuchung durchgeführt werden in der ggf. weitere Schritte abgeklärt werden. Findet ein operativer Eingriff, ohne therapeutische Begleitung statt, muss in jedem Fall eine Nachbehandlung durchgeführt werden. In einem leichteren Fall könnte es auch reichen eine therapeutische Begleitung durchzuführen. Diese Möglichkeit kann auch im Entscheider ausgeschlossen werden, so dass nur die Ausführungen „minimal“, „minimal mit Begleitung“, „operativ“, „operativ mit Begleitung“ möglich sind.

Für Petrinetze existieren die folgenden drei Bedingungen für Soundnes (vgl. [102, S. 274])

- Für jeden von einem Zustand i aus erreichbaren Zustand in der Gesamtmenge der Zustände, gibt es einen Weg der zum Endzustand führt.
- Der Endzustand ist der einzige Zustand, der „aktivierbar“ ist, wenn der von i aus erreicht ist. (d.h. der Workflow ist an keiner anderen Stelle mehr aktiv, sobald das Ende des Workflows erreicht wurde.)
- Es gibt keine Wege von einem Zustand i zum Endpunkt, die nicht erreicht werden können.

Diese Bedingungen lassen sich in Petrinetzen mit Hilfe von Erreichbarkeitsanalysen lösen (vgl. [102, S. 274]). Der dritte Punkt im obigen Beispiel erweist sich jedoch als schwierig. Dies ergibt sich daraus, dass unter der Voraussetzung sinnvoller Bedingungen und Entscheidungen, sich keine toten Pfade konstruieren lassen. Somit ist die strukturelle Korrektheit trivialerweise gegeben, da die Komposition auf AND-Splits und OR-Joins basiert.

Im folgenden Abschnitt werden weitere Abstufungen von Korrektheit beschrieben, die an Geschäftsprozesse angepasst sind: Relaxed-, Weak- und Lazy-Soundness (vgl. [102, S. 299f]).

Sei $PN = (P, T, F)$ ein Workflow-Netz:

- P1: $\forall M \left(i \xrightarrow{*} M \right) \Rightarrow \left(M \xrightarrow{*} o \right)$
Terminierung: Jeder Zustand, der aus dem Anfangszustand erreichbar ist, gelangt schließlich in den Endzustand.

- P2: $\forall M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow M = o$
Saubere Terminierung: Im Endzustand ist keine weitere Zustandsänderung mehr möglich. (Bei Petrinetzen bedeutet dies, dass die einzige Marke im Endzustand liegt.)
- P3: $(\forall t \in T) \exists M, M' : i \xrightarrow{*} M \xrightarrow{t} M'$
Keine toten Transaktionen: Jede Transition kann an dem Workflow teilnehmen.
- P4: $(\forall t \in T) \exists M, M' : i \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} o$
Keine Einbahnstrassen: Eine Transaktion darf nicht dazu führen, dass der Endzustand des Workflows nicht mehr erreicht werden kann. (Jede Transaktion muss in einer Konfigurationsfolge, die den Endzustand erreicht, enthalten sein.)

Daraus ergeben sich folgende Definitionen für Korrektheit:

- Soundness: $P1 \wedge P2 \wedge P3$
- Weak Soundness: $P1 \wedge P2$
- Relaxed Soundness: $P4$
- Lazy Soundness: $P1$

Diese Korrektheitsdefinitionen beziehen sich allerdings nur auf Workflow-Netze. Um sie auf das DynamicFlow-Konzept zu übertragen, könnte es formal in Petrinetze transformiert und die Kriterien anhand der Petrinetze zu bewiesen werden. Bei einer Beschränkung auf die Kernaussagen der Kriterien, sind die meisten auch ohne Transformation in Petrinetze einsichtig.

- P1: Jede Konfigurationsfolge endet im Endzustand
- P2: Im Endzustand darf keine weitere Konfigurationsfolge erreicht werden. Das bedeutet, kein Aktionsplan/Entscheider darf mehr aktiviert sein, da nur Aktionspläne und Entscheider aus Sicht des Workflows dessen Konfiguration verändern.
- P3: Zu jedem Aktionsplan gibt es eine Konfigurationsfolge, in der dieser aktiviert wird.
- P4: Die Aktivierung eines Aktionsplans/Entscheiders darf nicht dazu führen, dass das Ende des Workflows nicht mehr erreicht werden kann.

Bei der Betrachtung der Endbedingung des Konzeptes (d. h. es sind keine Aktionspläne bzw. Entscheider mehr aktiv (vgl. 3.3.9)), ist die Feststellung das P1 und P2 gelten offensichtlich. Wird von den Vorbedingungen abstrahiert und diese als erfüllbar vorausgesetzt (aus fachlicher Sicht sollte dies in medizinischen Leitlinien bzw. klinischen Pfaden der Fall sein), ist auch P3 erfüllt, da aus der strikten Benutzung von AND-Split und OR-Join die

Aktivierung jedes Teilpfades potentiell möglich ist. Somit bleibt festzustellen, dass P1 und P2 in diesem Konzept, aufgrund der Definition der Elemente grundsätzlich gegeben sind. Kriterium P3 hängt von dem fachlichen Inhalt der formulierten Regeln ab. Folglich ist Weak Soundness per Definition gegeben und es wird ferner davon ausgegangen, dass die verwendeten Regeln fachlich korrekt gesetzt wurden, gilt ebenfalls Soundness und die Korrektheit kann letztlich sichergestellt werden.

Das Or-Join Problem und die Auswirkungen auf die Korrektheit

Die Herangehensweise an das OR-Join Problem (Vgl. Kapitel 4.3.5) hat ebenfalls einen Einfluss auf die Soundness-Kriterien. Der Unterschied des fachlich gewollten OR-Joins gegenüber der technischen Umsetzung in BPEL führt dazu, dass die Ausführungssemantik und somit die Soundness-Eigenschaft ebenfalls verändert wird.

Ein OR-Join bedeutet aus fachlicher Sicht, dass ein nachfolgender Aktionsplan auch dann angestoßen wird, wenn nur einer der Vorgänger abgeschlossen ist. Andere Vorgänger könnten in Nachhinein abgeschlossen werden, oder sich in einer Konfigurationsfolge befinden, die nie erreicht wird, was unter Umständen beabsichtigt ist. Konnte beim ursprünglichen Konzept davon ausgegangen werden, dass beim OR-Join keine Deadlocks auftreten können, ist dies bei dieser Lösung möglich.

Reichte es vorher aus, festzustellen das ein Aktionsplan (oder ein ähnliches Element) abgeschlossen worden ist, muss nun zusätzlich festgestellt werden, dass ein Element nicht ausgeführt wird, und diese Information darüber hinaus an die nachfolgenden Elemente weiter gegeben werden.

3.4 Beschreibung des erhobenen klinischen Pfades

Dieses Kapitel beschreibt die Modellierung eines klinischen Pfades zur ambulanten Knie-OP mit Konzepten des DynamicFlow-Metamodells. Der Pfad wurde im Rahmen des Wisa-Projekts „Servicebasierte, prozessorientierte Orchestrierungs-Technologie“ (SPOT) [6] des Fraunhofer ISST und seiner Projektpartner erhoben. Als Beispiel hierzu dient der Teilpfad LA03 der Voruntersuchung.

Das Ziel von LA03 ist „die Untersuchung des Patienten, um Daten für eine Entscheidung zur Durchführung einer ambulanten Operation zu erheben“. Hierzu werden Anamnese und Sozialanamnese durchgeführt. Außerdem wird über die Notwendigkeit einer radiologischen Untersuchung entschieden. Es wird zudem eine ausreichende Beratungstätigkeit erbracht, um Fragen des Patienten zu beantworten und mit ihm das weitere Procedere abzusprechen“ [66]. Eine gesamte Übersicht über die Prozedurbeschreibung zur Leistung LA03 ist [66] S.12 zu entnehmen.

3.4.1 Umsetzung des Pfades im Rahmen der PG

Der entstehende Workflow wird in Abbildung 3.14 dargestellt. Im Workflow werden Aktionspläne, die übersprungen werden können, speziell gekennzeichnet (z. B. AP02).

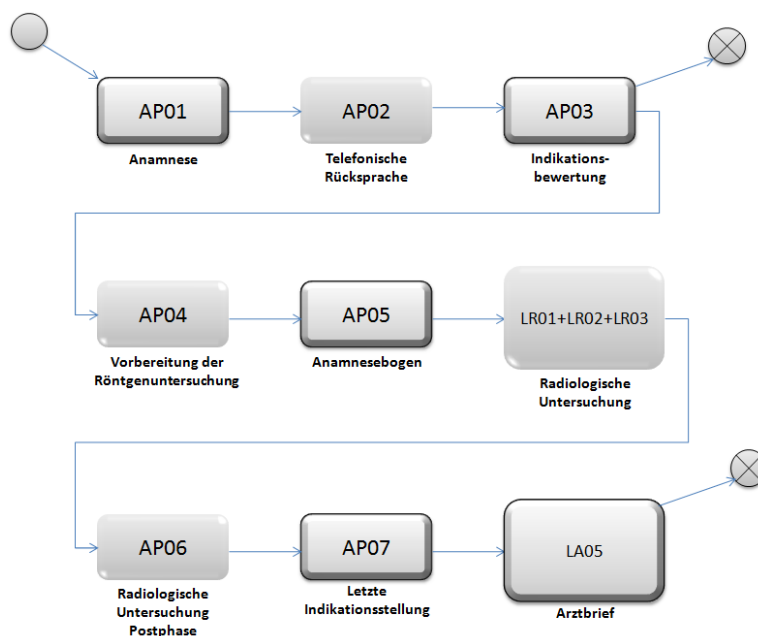


Abbildung 3.14: Workflow zur Leistung LA03.

Im Folgenden wird kurz erläutert, welche Teile der Prozedurbeschreibung der Leistung LA03 zu welchen Aktionsplänen zusammengefasst wurden.

Die Abbildung 3.15 stellt dar, wie die ersten Teile der Prozedurbeschreibung, vom *Aufruf des Patienten* bis zur *Bestimmung der Indikationsstellung*, im Aktionsplan AP01 zusammengefasst werden. Die Notation erfolgt in der „SPOT Modelling Language“ (SPOT ML) [19]. Der Aufruf des Patienten und der Patientendaten könnte zwar selbst je einen Aktionsplan darstellen, sie werden hier jedoch als Vorbereitungsschritte zur Durchführung der Anamnese gezählt und damit als Teil der Instruktionen des Aktionsplans *Anamnese* betrachtet.

Nachdem das System die Indikationsstellung vorgeschlagen hat, kann der Arzt bei Bedarf eine *telefonische Rücksprache* vornehmen. Da sie optional ist, wird sie durch einen Aktionsplan (siehe Abb. 3.16) repräsentiert, damit es möglich wird, sie mit Hilfe einer SKIP-Regel überspringen zu können.

Der Arzt muss nun die vom System vorgeschlagene Indikationsstellung bewerten und im Falle einer Abweichung die Gründe in Form einer Varianzdokumentation festhalten. Diese Aktionen werden zu einem Aktionsplan zusammengefasst (siehe Abb. 3.17).

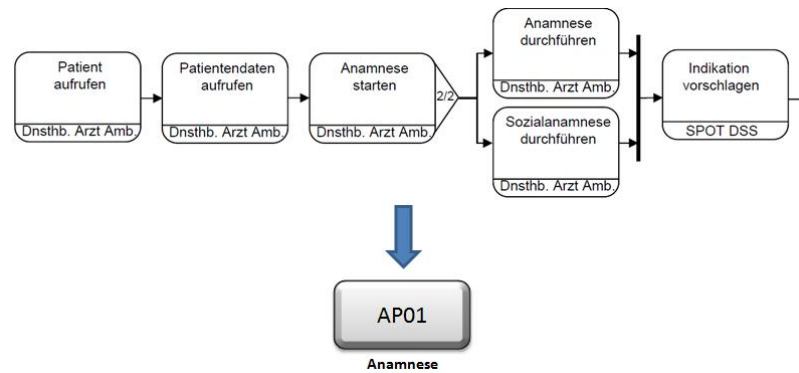


Abbildung 3.15: Erste Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

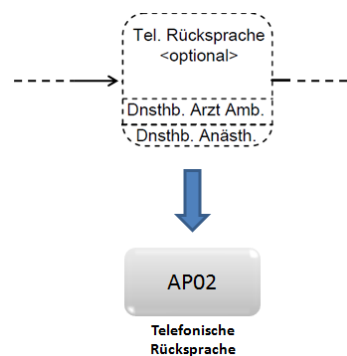


Abbildung 3.16: Zweite Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

Die Entscheidung zur Feststellung der gegebenen Indikationsstellung oder der Notwendigkeit einer Röntgenuntersuchung werden im DynamicFlow-Metamodell durch Abbruchregeln bzw. Übersprungregeln repräsentiert. Zusammen mit den Schritten der Vorbereitungsphase der Röntgenuntersuchung bilden sie einen Aktionsplan (siehe Abb. 3.18), welcher sowohl eine *ABORT*-Regel als auch eine *SKIP*-Regel enthält. Auf diese Weise wird die Möglichkeit zum Überspringen von Varianzdokumentation und Röntgenuntersuchung modelliert.

Anschließend wird die nicht optionale Durchführung der Anamnese vorgenommen. Nachdem der Anamnesebogen ausgedruckt und vom Patienten unterschrieben wurde, legt der Arzt den Bogen in der Patientenmappe ab. Dies wird in Form des Aktionsplans AP05 zusammengefasst (siehe Abb. 3.19).

Falls angeordnet wird im weiteren Verlauf der Behandlung eine radiologische Untersuchung durchgeführt, welche nicht mehr Bestandteil der Leistung LA03 ist. Nach der Untersuchung meldet sich der Patient wieder an der Ambulanz an. Sobald der Kurzbefund durch den Radiologen erstellt wurde und elektronisch im System zur Verfügung steht, wird der Patient vom Arzt ein weiteres Mal zur Sprechstunde aufgerufen. Anmeldung und Aufruf können in

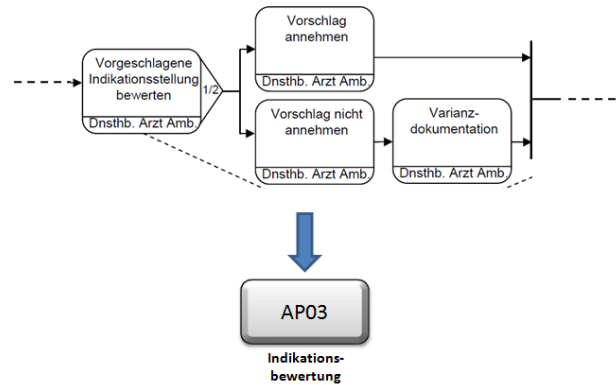


Abbildung 3.17: Dritte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

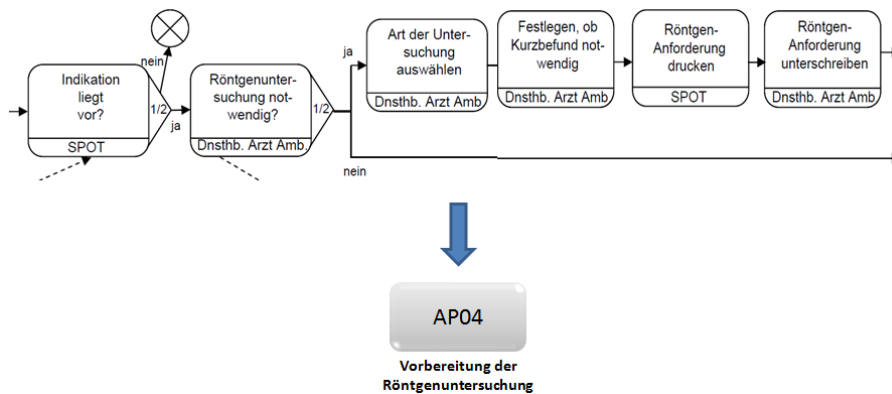


Abbildung 3.18: Vierte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

einem Aktionsplan zusammengefasst werden (siehe Abb. 3.20). Die Aktion *Warten auf den Kurzbefund* wird dabei durch eine *WAIT*-Regel abgebildet.

Die drei letzten Teile der Prozedurbeschreibung, vom Aufruf der Patientendaten bis zur endgültigen Indikationsstellung, werden stets entweder nach Ausführung der *radiologischen Untersuchung Postphase* oder direkt nach *Ablegen des Anamnesebogens in der Patientenmappe* in Form des Aktionsplans AP07 zusammengefasst (siehe Abb. 3.21). Im Falle einer negativen Entscheidung muss eine Dokumentation vom Arzt erstellt werden. Auf jeden Fall folgt danach die Leistung LA05 zur Erstellung des Arztbriefs.

Ein besonderer Punkt, der aus der in [66] dargestellten Abbildung 3.22 entnommen werden kann, ist das Vorkommen von Zyklen im Prozessmodell. Wie in der Prozedurbeschreibung zur Leistung LA05 zu sehen ist, könnte der Arztbrief nach einer Prüfung in iterativer Weise zur Korrektur vorgelegt werden. Eine Idee, dies umgehen zu können, wäre es, den sich wiederholenden Schleifenrumpf statt als Instruktionen in Form von einem Aktionsplan zu

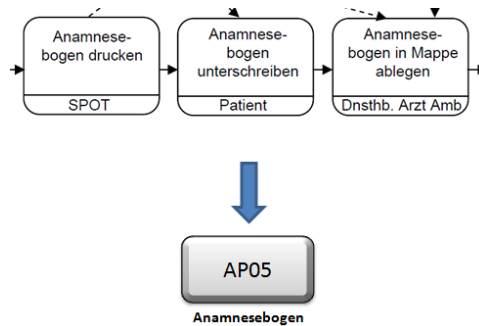


Abbildung 3.19: Fünfte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

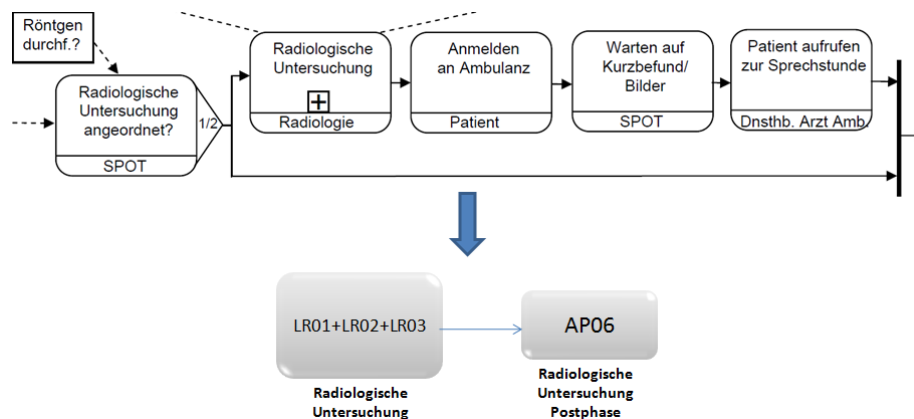


Abbildung 3.20: Sechste Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

modellieren. Dadurch wäre es möglich, bei jeder Iteration den Aktionsplan im Ad-hoc-Modus erneut einzufügen und damit wird die Schleife schrittweise „abgerollt“.

Im Folgenden werden die Aktionspläne genauer beschrieben. Die Ablaufbedingungen (siehe Anhang A.1), Instruktionen mit den konsumierten und erzeugten Daten (siehe Anhang A.2) und die dafür verwendete Variablen (siehe Anhang A.3) werden durch Tabellen bzw. Listen im Anhang zusammengefasst.

Der erste Aktionsplan des Workflows ist AP01 *Anamnese*. Es wird darauf gewartet, dass die Variablen *pav* und *usv* (siehe Anhang A.3 für eine komplette Auflistung und genauere Beschreibung der Variablen) mit dem Wert *true* belegt werden, damit der Aktionsplan aktiviert wird. Nach der Aktivierung werden fünf Instruktionen durchgeführt, wobei die Abhängigkeiten zwischen den einzelnen Instruktionen im Anhang A.2 zu finden sind. Der Patient wird hier von einem bestimmten Arzt aufgerufen und von einem Wartezimmer der Ambulanz in einem zuvor festgelegten Untersuchungsraum eingeladen, wo alle Untersuchungen des Tages durchgeführt werden. Da die Patientenakte schon vorhanden ist, kann der Arzt den

3.4 Beschreibung des erhobenen klinischen Pfades

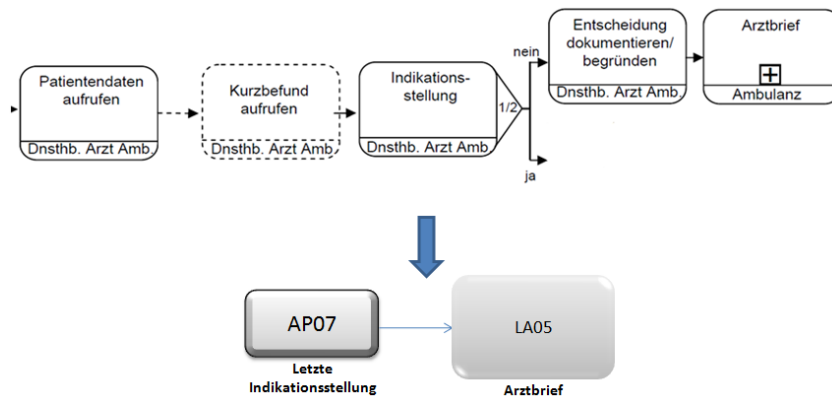


Abbildung 3.21: Siebte Teilumsetzung der Prozedurbeschreibung zur Leistung LA03 [66].

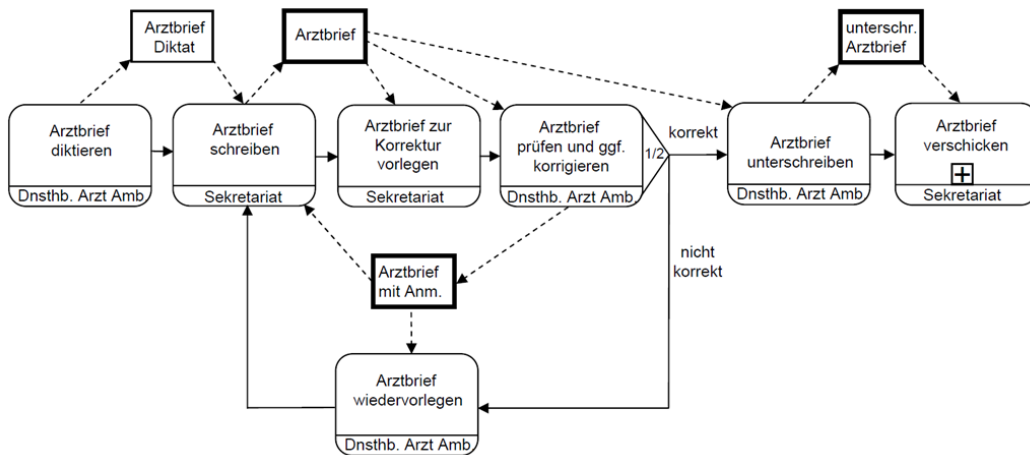


Abbildung 3.22: Prozedurbeschreibung zur Leistung LA05 Arztbrief [66].

Patientendatensatz auf seinem Endgerät (z. B. Tablet-PC) aufrufen. Die dritte bzw. vierte Instruktion zeigen dem Arzt ein Formular an und speichern es nach Bearbeitung, wobei der Arzt dem Patienten Fragen zu Beschwerden, Vorerkrankungen und medikamentöser Behandlung bzw. zu den sozialen Aspekten stellt. Mit der letzten Instruktion schlägt das System dem Arzt anhand mehrerer Kriterien eine Indikation vor. Im Falle einer vorliegenden chronischen Erkrankung oder einem Patientenalter jünger als drei Monate, schlägt das System z. B. vor, eine anästhesiologische Konsultation vorzunehmen. Im Falle einer positiven Indikation nimmt die Variable *indKnieOpSYS* (siehe Anhang A.3) den Wert **true** an.

Der zweite Aktionsplan erfasst die *telefonische Rücksprache* und kann durch die Bedingung $\neg ce \wedge (alt > 3Monate)$ (siehe Anhang A.3) übersprungen werden, d. h. wenn der Patient nicht chronisch erkrankt ist und älter als drei Monate ist. Der Aktionsplan wird hier aktiviert, wenn eine Indikationsbewertung schon vom Experten-System angeschlagen wurde.

Der diensthabende Arzt der Ambulanz plant eine Rücksprache mit einem bestimmten Anästhesist, ruft diesen ggf. an und erstellt eine Notiz zum Gespräch.

Der dritte Aktionsplan AP03 ist die *Indikationsbewertung*. Er wird aktiviert, wenn die Bedingung $(indKnieOpSYS \wedge pav \wedge abv \wedge sabv)$ (siehe Anhang A.3) erfüllt ist, also wenn sowohl die Indikationsbewertung des Experten-Systems, die Patientenakte als auch der Anamnese- und der Sozialanamnesebogen vorhanden sind. Erst nach der Aktivierung werden durch die drei ersten Instruktionen Dokumente bereitgestellt. Hierbei ruft der Arzt den Patientendatensatz, sowie den Fragebogen zur Anamnese und zur Sozialanamnese auf seinem Endgerät auf. Anhand dieser Dokumente erfolgt eine Bewertung, ob eine ambulante Durchführung der OP möglich ist. Das Ergebnis der Indikation wird in der Variable *indKnieOpDA* (siehe Anhang A.3) gespeichert. Anschließend wird vom behandelnden Arzt eine Abweichungsdokumentation erstellt, indem hier die zwei Variablen *indKnieOpDA* und *indKnieOpSYS* verglichen werden und festgestellt wird, ob sie ungleich sind. Die Abweichungsdokumentation wird z. B. in einem Dokument gespeichert. Im Falle einer positiven Indikation nimmt die Variable *indKnieOpDA* den Wert **true** an.

Der Aktionsplan AP04, die *Vorbereitung der Röntgenuntersuchung*, könnte aufgrund der Nicht-Erfüllung der Bedingung $roentVorh \vee befVorh$ (siehe Anhang A.3) übersprungen werden, falls die Röntgenbilder und der Befund des Patienten nicht vorhanden sind. Falls die Variable *indKnieOpDA* zu dem Wert **false** ausgewertet wurde, wird der Workflow abgebrochen. AP04 enthält vier Instruktionen. Mit der ersten wird ausgewählt, welche Art der Untersuchung aus einer Menge von Untersuchungsarten (z. B. Röntgen, MRT, etc.) durchzuführen ist. Die erzeugte Variable *art* bestimmt die ausgewählte Art. Im nächsten Schritt wird der Befundtyp festgelegt. Hier reicht es entweder einen normalen Befund zu erstellen oder es muss zusätzlich ein Kurzbefund erstellt werden. Anhand der Daten aus dem Patientendatensatz, der Untersuchungsart und dem Befundtyp wird automatisch ein Röntgenschein erstellt und mit dem Drucker des Untersuchungsraums ausgedruckt. Der so erstellte Röntgenschein wird vom Arzt unterzeichnet, womit die Variable *roentSchU* (siehe Anhang A.3) den Wert **true** annimmt.

Der fünfte Aktionsplan AP05 der *Anamnesebogen* wird nun aktiviert. Er enthält drei Instruktionen. Zunächst wird anhand der Anamnesedaten automatisch ein Anamnesebogen erstellt und mit dem Drucker des Untersuchungsraums ausgedruckt. Des Weiteren wird der Patient veranlasst, den Anamnesebogen zu unterschreiben, was zur Folge hat, dass die Variable *anamBogU* (siehe Anhang A.3) den Wert **true** annimmt. Letztendlich wird der unterschriebene Ausdruck des Anamnesebogens vom Arzt in der präoperativen Mappe abgelegt.

Der Aktionsplan AP06 *Radiologische Untersuchung Postphase* erfolgt nach der Leistung *Radiologische Untersuchung* und könnte übersprungen werden, wenn die Bedingung $\neg roentSchU$ (siehe Anhang A.3) erfüllt ist, d. h. der Arzt hat keine Röntgenuntersuchung angeordnet.

Der Aktionsplan wird nur aktiviert, wenn die Bedingung $roentVorh \vee befVorh$ (siehe Anhang A.3) erfüllt wird, d. h. wenn die Röntgenbilder bzw. der Befund nach der radiologischen Untersuchung erstellt wurden. Nach der Aktivierung meldet sich der Patient wieder an der Ambulanz an, um vom Arzt ein weiteres Mal zur Sprechstunde aufgerufen zu werden.

Der Aktionsplan AP07 behandelt die *letzte Indikationsstellung*. Hier werden nochmals alle benötigten Dokumente, d. h. der Patientendatensatz, der Fragebogen zur Anamnese und zur Sozialanamnese bereitgestellt. Außerdem ruft der Arzt den Befund auf seinem Endgerät auf. Sobald der Arzt den Befund aufruft, prüft das System durch die Variable *kurz*, ob es einen Kurzbefund gibt. Ist dies der Fall, ruft das System automatisch den Kurzbefund auf. Nun bewertet der Arzt nochmals die Indikationsstellung anhand der bereitgestellten Dokumente. Das Ergebnis der Indikation überschreibt die Variable *indKnieOpDA*. Falls die Indikation negativ ist, erstellt der Arzt ein Entscheidungsdokument, indem er seine Entscheidung anhand geeigneter Multiple-Choice-Antworten begründet (siehe [66]). Somit ist die Beschreibung der Leistung LA03 beendet. Mit der Erstellung des Arztbriefs erfolgt der Abschluss des Workflows. Falls die Indikation positiv ist, folgen die Leistungen LA04 (organisatorische OP-Vorbereitung) und LA05 (Arztbrief zur OP-Vorbereitung).

4 Technologie

Nachdem das Konzept theoretisch beschrieben wurde, wird in diesem Kapitel hergeleitet, wie das Konzept in der Business Process Execution Language (BPEL) realisiert werden kann. Somit wird es möglich, dass modellierte Prozesse automatisiert ausgeführt werden. Zunächst wird die Entstehungsgeschichte von BPEL vorgestellt und anschließend erklärt, wie einzelne Komponenten des Konzeptes in Sprachelemente von BPEL umgesetzt werden.

4.1 Ausführungssprache BPEL

In diesem Kapitel erfolgt zunächst eine kurze Einführung in die zur Umsetzung der beschriebenen Konzepte gewählte Ausführungssprache BPEL und der darauf aufbauenden Sprache BPEL4People (Kapitel 4.2), die zur Umsetzung menschlicher Aktivitäten dient. Anschließend wird in Kapitel 4.3 die Transformation von Workflows, Aktionsplänen und Instruktionen in BPEL beschrieben.

4.1.1 Geschichte

Microsoft hat im Jahr 2000 damit begonnen die XLANG-Spezifikation zu entwickeln. Bei XLANG handelt es sich um eine blockorientierte Sprache, welche als Programmiersprache für den Microsoft BizTalk Server gedacht war. Das Ziel der Sprache ist die Beschreibung von Geschäftsprozessen und Interaktionen zwischen „Web Service Providern“. Dabei wurde als Mittel zur Beschreibung der Prozess-Service-Schnittstelle die Beschreibungssprache „Web Services Description Language“ (WSDL) genutzt.

Parallel dazu folgte IBM mit der graphorientierten Workflow Sprache WSFL (Web Service Flow Language) im Jahr 2001. WSFL definiert ein Modell von Aktivitäten, die in einem Graph durch Kontroll- und Datenflüsse verbunden sind.

In 2002 haben sich die beiden Unternehmen entschieden, die beiden Lösungen der Block- und Graphstrukturierung zu kombinieren. BEA Systems, ebenfalls ein weltweit tätiges Softwareunternehmen, hat bei der Entwicklung mitgewirkt. Das Ergebnis der Entwicklung war BPEL4WS (Business Process Execution Language for Web Services), welches im Juli 2002 veröffentlicht wurde. Im weiteren Verlauf haben SAP und Siebel weiter an der Spezifikation mitgearbeitet und gemeinsam entstand BPEL 1.1. Im Mai 2003 wurde BPEL 1.1 bei dem technischen Komitee der OASIS eingereicht, so dass die Spezifikation als offizieller Standard weiter entwickelt werden konnte. Das Ergebnis der OASIS-Entwicklung war die Ausgabe von WS-BPEL 2.0 und wurde im April 2007 als OASIS Standard verabschiedet. [40] [65]

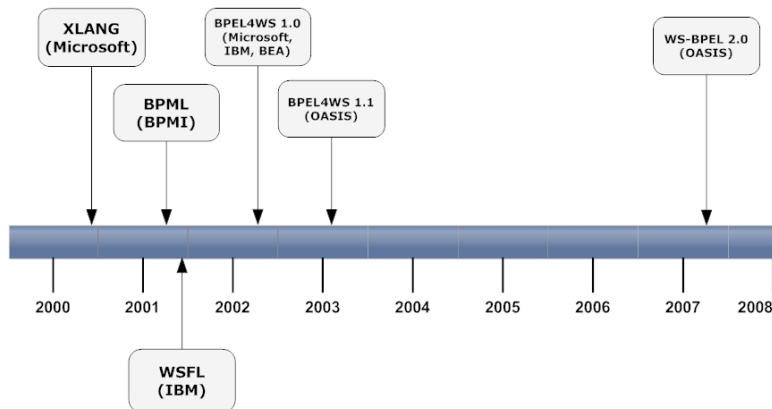


Abbildung 4.1: Entwicklungsfortschritt: von XLANG bis WS-BPEL 2.0 .

4.1.2 BPEL vs. BPML

Ein ernst zunehmender Konkurrent von BPEL war BPML (Business Process Modelling Language), welches von BPMI.org (Business Process Management Initiative) entwickelt wurde und wie BPEL eine Orchestrierungssprache ist (aus [40]).

Wenn BPML mit BPEL verglichen wird, stellt sich heraus, dass beide Sprachen die gleichen Web Service Basisspezifikationen, als auch besondere Spezifikationen wie WS-Sicherheit, WS-Koordination und WS-Transaktionen unterstützen. Dank Unterstützung erweiterter Semantiken wie „nested Processes“ wird in BPML jedoch die Modellierung komplexerer Geschäftsprozesse unterstützt.

Der Grund, dass BPML nicht als Standard anerkannt wurde, war, dass die Version 1.0 einen Monat nach der ersten Version von BPEL erschien und BPEL durch Unternehmen wie Microsoft, SAP und IBM unterstützt wurde. Folglich konnte BPML im Standard-Rennen nicht mehr mithalten und BPMI.org erkannte letztendlich an, dass BPEL die am besten unterstützte Version sein würde. [40] [65]

4.1.3 BPEL Konzepte

BPEL ist eine Sprache zur Orchestrierung von Web Services und somit wird eine völlige Automatisierung der Geschäftsprozesse ermöglicht.

4.1.3.1 Einführung in BPEL

Die Hauptmerkmale der Sprache sind:

- **XML-basierte Prozessbeschreibungssprache:** BPEL ist komplett in XML ausgedrückt und nutzt die WSDL und XML Schema (XSD) für das Datenmodell. Weiterhin

ist in BPEL der Standard XPATH integriert, der zum Formulieren von Ausdrücken und Anfragen verwendet werden kann.

- **Bedeutung von Web Services in BPEL:** Web Services sind sowohl die Prozesseingabe als auch Prozessausgabe. BPEL Prozesse sind als Services implementiert, weil sie das WSDL Schnittstellen-Modell verfolgen. Kurz gefasst ist ein BPEL Prozess ein Web Service.
- **Orchestrierung zur Beschreibung von Web Services:** Der Hauptteil bei der Orchestrierung ist der sogenannte zentrale Koordinator, der für die Ausführung der Operationen von Web Services zuständig ist. Dabei „wissen“ die beteiligten Web Services selbst nicht, ob sie in einem „higher-level“ Geschäftsprozess teilnehmen. Nur dem zentralen Koordinator ist „bewusst“, welche Web Services beteiligt sind und in welcher Reihenfolge sie aufgerufen werden sollen. Hier wird ein Orchestrierungsserver benötigt, um eine Orchestrierung in der Praxis zu realisieren. Dies wird üblicherweise in einem speziellen Applikationsserver eines Unternehmens integriert.
- **Unterstützung von ausführbaren sowie von abstrakten Prozessen:** In BPEL besteht ein Geschäftsprozess aus Elementen, die das Verhalten einer Aktivität definieren und einschließlich die Fähigkeit haben, Web Services und Kontrollfluss aufzurufen als auch zu „kompensieren“, falls Fehlern auftreten. Man unterscheidet zwischen zwei Arten von Geschäftsprozessen:
 - Ausführbare Geschäftsprozesse: Sie spezifizieren die exakten Details von Geschäftsprozessen und können durch eine BPEL-Engine ausgeführt werden.
 - Abstrakte Geschäftsprozesse: Hier wird nur der Nachrichtenaustausch spezifiziert und Details der Prozessflüsse nicht betrachtet. Abstrakte Prozesse sind nicht ausführbar und werden daher selten benutzt.
- **BPEL ist an sich ein Web Service:** Dank WSDL Schnittstelle können BPEL-Prozesse genau wie normale Web Services adressiert werden, und rekursive Konstruktionen von Geschäftsprozessen können beschrieben werden.

4.1.3.2 Hauptmerkmale von WS-BPEL 2.0

Anhand der vorher erwähnten Konzepte von BPEL ergeben sich u.a. die folgenden Merkmale der Spezifikation:

- **Prozesslebenszyklus und Prozesstypen:**

Ein BPEL Prozess hat typischerweise einen Status und kann eine langfristige Dauer haben. Ein Prozess wird durch eine empfangene Nachricht initialisiert. Wird der Lebenszyklus eines Prozesses beendet, so führt dies zu seiner Terminierung.

Wie schon erwähnt wurde, definiert BPEL abstrakte Prozesse, die zur Spezifizierung

von Geschäftsprotokollen verwendet werden können und ausführbare Prozesse, die einen Workflow implementieren. Hier wird, bei der Definition der Kernkonzepte durch die BPEL Spezifikation keine Unterscheidung zwischen abstrakten und ausführbaren Prozessen gemacht.

- **Partner-Links und Kommunikation:**

WSDL definiert Typen, Nachrichten, Operationen und Porttypen. Des Weiteren spezifiziert WSDL konkrete Bindungen und Services, d.h. wie und wo die definierten Operationen aufgerufen werden.

BPEL verwendet die abstrakten Konstrukte von WSDL und erweitert diese durch neue Elemente inklusive des `partnerLinkType`. Ein `partnerLinkType`, welcher in einer WSDL-Datei (die WSDL-Schnittstelle des Prozesses) definiert ist, beschreibt wie der Nachrichtenaustausch zwischen zwei WSDL-Schnittstellen aussehen soll. Der `partnerLinkType` charakterisiert diesen Austausch durch Definition von Rollen und durch die Spezifizierung des vom Service gelieferten `portType`.

- **Datenverarbeitung:**

Geschäftsprozesse bilden zustandsorientierte Interaktionen ab. Der beteiligte Zustand besteht sowohl aus gesendeten und empfangenen Nachrichten, als auch aus anderen relevanten Daten wie z. B. Zeitbeschränkungen (Timeouts). Um den Prozesszustand beizubehalten, werden in BPEL Zustandsvariablen (werden im Folgenden einfach Variablen genannt) verwendet. Außerdem stellt BPEL so genannte Ausdrücke zur Verwaltung der in den Variablen enthaltenen Daten zur Verfügung. Weiterhin werden Zuweisungen benötigt um Prozesszustände zu aktualisieren. Dies wird in BPEL durch die `<assign>`-Aktivität ermöglicht.

Variablen enthalten sowohl Nachrichten, die zwischen dem Prozess und seinen Partnern ausgetauscht werden, als auch Daten, die nur innerhalb des Prozesses anfallen, sogenannte „private Daten“. Der Typ der Variable wird durch eines der folgenden Attribute definiert:

- `messageType`: Variable kann eine WSDL-Nachricht enthalten
- `element`: Variable kann ein XML-Schema-Element enthalten
- `type`: Variable kann einen einfachen XML-Schema-Typ enthalten

Die folgenden Beispiele und Quelltextauszüge sind dem Loanapproval BPEL Prozess entnommen, der standardmäßig als Tutorial im ActiveBPEL-Designer enthalten und online verfügbar [7] ist. Der ActiveBPEL-Designer wird in Kapitel 5.1.2 eingehender beschrieben.

Listing 4.1: Variablendefinition

```
<variables>
```

```

<variable messageType="loandef:creditInformationMessage"
  name="request"/>
<variable messageType="asns:riskAssessmentMessage"
  name="riskAssessment"/>
<variable messageType="apns:approvalMessage"
  name="approvalInfo"/>
<variable messageType="loandef:loanRequestErrorMessage"
  name="error"/>
</variables>

```

Zuweisung: Die Zuweisung von Werten an Variablen erfolgt über die `<assign>`-Aktivität. Ein Beispiel für eine `<assign>`-Aktivität findet sich im folgenden Abschnitt unter „Atomare Aktivitäten“.

Ausdrücke: An vielen Stellen ist es nötig über Variablen z. B. logische Bedingungen, insbesondere im Zusammenhang mit der Steuerung von Kontrollflüssen, zu formulieren als auch Berechnungen ausführen zu können. Dafür werden in BPEL fünf Typen von Ausdrücken unterstützt: Boolesche- (mit Hilfe von `<transition>`, `<if>`, ...), Deadline- (mit Hilfe von `<WAIT>` und `<onAlarm>`), Laufzeit- (mit Hilfe von `<WAIT>` und `<onAlarm>`), vorzeichenlose Ganzzahl- (mit Hilfe von `<startCountValue>`) und allgemeine Ausdrücke (mit `<assign>`).

Das folgende Beispiel zeigt die Umsetzung eines booleschen Ausdrucks:

Listing 4.2: Boolescher Ausdruck

```

<source
  linkName="receive-to-approval"
  transitionCondition="bpws:getVariableData('request',
    'amount') >= 10000"
/>

```

- **CorrelationSet:**

Bei langlaufenden Geschäftsprozessen ist es notwendig, eintreffende Nachrichten an die richtige Prozessinstanz zu leiten. Diese Aufgabe übernimmt das „CorrelationSet“. Wenn z. B. eine Nachricht von einem Partner eingeht, ist es erforderlich zu entscheiden, ob ein neuer Prozess instanziiert werden soll oder diese Nachricht an eine existierende Prozessinstanz adressiert werden soll. Statt dafür das Konzept von Instanz-IDs zu benutzen, werden in BPEL Informationen (z. B. Patient-ID) wiederverwendet, die durch Markierungen in eingehenden Nachrichten identifiziert werden können.

- **Atomare Aktivitäten (basic activities):**

Eine BPEL-Prozessbeschreibung definiert einen Workflow bestehend aus einer Reihe von Arbeitsschritten. Jeder Schritt in einem Workflow ist durch atomare und strukturierte Aktivitäten implementiert. Jede atomare Aktivität hat verschiedene Standar-

dattribute und -elemente, die benutzt werden können, um bestimmte Eigenschaften zu spezifizieren.

Assign: Die *<assign>*-Aktivität besteht aus mehreren Zuweisungen. Jede Zuweisung wird durch eine Kopie eines Elements von einem Attribut zum anderen definiert. Die Kopiequelle (wird durch „from“ spezifiziert) und das Kopieziel (wird durch „to“ spezifiziert) müssen typkompatibel sein.

Das folgende Beispiel illustriert eine Zuweisung:

Listing 4.3: *<assign>*-Aktivität

```
<bpel:assign name="Assign1">
  <bpel:copy>
    <bpel:from>
      <bpel:literal>1</bpel:literal>
    </bpel:from>
    <bpel:to variable="NewOperationResponse"/>
  </bpel:copy>
</bpel:assign>
```

Invoke: Mit einer *<invoke>*-Aktivität kann ein Web Service, der als Partner definiert wurde, aufgerufen werden. Der Aufruf kann synchron oder asynchron erfolgen.

Listing 4.4: *<invoke>*-Aktivität

```
<invoke inputVariable="request"
  name="invokeapprover"
  operation="approve"
  outputVariable="approvalInfo"
  partnerLink="approver"
  portType="apns:loanApprovalPT">
</invoke>
```

Receive: Eine *<receive>*-Aktivität spezifiziert einen partnerLink, einen PortTyp und eine Operation, die aufgerufen werden können.

Listing 4.5: *<receive>*-Aktivität

```
<receive createInstance="yes" name="ReceiveCustomerOrder"
  operation="SubmitOrder" partnerLink="Customer"
  variable="SubmitOrderRequest">
  <correlations>
    <correlation initiate="yes" set="CustomerCorrelationSet"/>
  </correlations>
</receive>
```

Reply: Eine *<reply>*-Aktivität wird nur bei einer synchronen Interaktion genutzt, um eine Antwort zu senden, nachdem eine Receive-Aktivität aufgerufen wurde.

Listing 4.6: <reply>-Aktivität

```
<reply
  name="reply"
  operation="approve"
  partnerLink="customer"
  portType="apns:loanApprovalPT"
  variable="approvalInfo">
</reply>
```

Throw: Wenn ein Prozess über einen internen Fehler berichten will, wird dafür die <throw>-Aktivität verwendet.

Listing 4.7: <throw>-Aktivität

```
<throw faultName="rws:CreditCheckFault"
  faultVariable="CreditCheckFault"
  name="ThrowCreditCheckFault"/>
```

WAIT: Eine <WAIT>-Aktivität bietet die Möglichkeit auf einem Zeitpunkt oder für eine Zeitspanne zu warten. Im folgenden Beispiel aus dem ActiveBPEL Benutzerhandbuch [8], muss der Prozess für eine Dauer von 1 Jahr, 2 Monate, 3 Tage, 10 Stunden und 30 Minuten warten.

Listing 4.8: <WAIT>-Aktivität

```
<WAIT for=" 'P1Y2M3DT10H30M' "/>
```

Empty: Die <empty>-Aktivität kann als Platzhalter für spätere Implementierungen dienen oder kann genutzt werden, falls an einer bestimmten Stelle im Prozess keinerlei Aktivität ausgeführt werden soll. Dies könnte z. B. bei der Fehlerbehandlung der Fall sein, um auftretenden Fehler zu unterdrücken. Die Syntax dieser Aktivität wird wie folgt illustriert:

Listing 4.9: <empty>-Aktivität

```
<empty name="nothingToDo" />
```

Exit: die <exit>-Aktivität beendet sofort die Prozessinstanz ohne jegliche Terminierungs- oder Fehlerbehandlung zu berücksichtigen.

- **Strukturierte Aktivitäten (structured activities)**

Strukturierte Aktivitäten beschreiben die Reihenfolge, in welcher eine Menge von Aktivitäten ausgeführt wird.

- Sequentielle Kontrolle zwischen Aktivitäten wird durch <sequence>, <if>, <while>, <repeatUntil> und die serielle Variante von <forEach> unterstützt.

- Nebenläufigkeit und Synchronisation von Aktivitäten wird durch `<flow>` und die parallele Variante von `<forEach>` unterstützt. Ein Beispiel zu Flow aus dem ActiveBPEL Benutzerhandbuch [9], das zwei parallele Receive Aktivitäten gleichzeitig startet, zeigt Listing 4.10

Listing 4.10: `<flow>`-Aktivität

```

<flow name="MarketplaceFlow">
  <receive name="SellerReceive" partnerLink="seller"
    portType="tns:sellerPT" operation="submit"
    variable="sellerInfo" createInstance="yes">
  </receive>
  <receive name="BuyerReceive" partnerLink="buyer"
    portType="tns:buyerPT" operation="submit"
    variable="buyerInfo" createInstance="yes">
  </receive>
</flow>

```

- Auf nicht-deterministisch eintreffende, externe Ereignisse kann mit der `<pick>`-Aktivität reagiert werden. Hierbei wird auf das Eintreffen einer klar definierten Nachricht gewartet. Optional kann die Zeitangabe in einer `<onAlarm>`-Aktivität genutzt werden, um dafür zu sorgen, dass eine bestimmte Reaktion nach Ablauf der angegebenen Zeit ausgelöst wird.

• Scopes und Handler:

Scopes ermöglichen die Verkapselung der Logik zusammen mit Kompensations-, Fehler-, Terminierungs-, Ereignishandler und lokalen Variablen.

Kompensationshandler: Dadurch wird eine Menge von Aktivitäten definiert, die ausgeführt werden müssen, falls ein Problem im Prozess auftritt. Der Handler kann aus dem Prozess selbst gestartet werden, um die schon vorher abgeschlossenen Schritte rückgängig zu machen.

Faulthandler: Ein Faulthandler kann optional im Scope eingebaut werden. Er definiert eine Menge von Fehlerbehandlungsaktivitäten, die `<catch>`-Aktivitäten genannt werden. Eine `<catch>`-Aktivität fängt einen Fehler durch seinen Fehlernamen und seine Fehlervariable. Tritt ein Fehler ohne Fehlernamen auf, dann wird der Fehler jedem Fehlerbehandler, der den gleichen Typ der Fehlervariablen hat, zugeordnet. Ein Beispiel zum Faulthandler aus dem Loanapproval Prozess sieht folgendermaßen aus.

Listing 4.11: Faulthandler

```

<faultHandlers>
  <catch faultName="apns:loanProcessFault"
    faultVariable="error">
  </catch>
</faultHandlers>

```

Terminierungshandler: Hier wird der Handler für Scope benötigt, um eine gezwungene Terminierung anzuwenden. Der Terminierungshandler ist aktiv solange das Scope beim Ausführen seine Hauptaktivität oder sein Ereignishandler ist.

Ereignishandler: Der Handler gibt vor „was zu tun“ ist, wenn bestimmte Ereignisse auftreten. Nachrichteneignisse können im Handler mehrmals sogar nebenläufig auftreten, solange das Scope aktiv ist. Im Nebenläufigkeitsfall werden Maßnahmen wie Serialisieren von Scopes getroffen. Ereignishandler werden typischerweise verwendet, um Abbruchnachrichten zu behandeln.

4.2 BPEL4People

Bei vielen Geschäftsprozessen ist die Interaktion mit menschlichen Bearbeitern notwendig. Hier kann BPEL durch die Erweiterung BPEL4People helfen die Interaktion und Kommunikation mit dem Benutzer zu steuern. Einerseits durch ausführbare Geschäftsprozesse, andererseits durch erkennbare Verwaltung von Web Service. Mit Hilfe von BPEL können auch Kontrollflüsse komplexer Prozesse abgebildet werden, wie z. B. die Fehler- und die Ereignisbehandlung. Die BPEL-Spezifikation konzentriert sich auf die Geschäftsprozesse, deren Aktivitäten als Web Service ohne zusätzliche Benutzerinteraktion modelliert werden können. Die Benutzerinteraktionen werden aus einem einfachen Szenario in ein komplexeres übertragen, wobei nicht nur die manuelle Zustimmung, sondern auch eine Dateneingabe durch den Benutzer abgebildet werden kann. An dieser Stelle ist die Vergabe eines Kredites als Beispielprozess vorstellbar. Auf dem Internetportal der Bank wird dem Interessenten eine grafische Benutzerschnittstelle angeboten. Hier werden mögliche Optionen und Konditionen angeboten und der Benutzer kann seine Wünsche in entsprechende Formulare eintragen. Das Absenden des ausgefüllten Onlineformulars startet einen internen Prüfprozess, der unter bestimmten Bedingungen, wie z. B. „die Authentizität des Benutzers ist bestätigt“, Kenntnis über die Vermögensverhältnisse und Sicherheiten, dem Kunden direkt eine Zu- oder Absage erteilen kann. In diesem Fall läuft der Prozess ohne weitere Interaktion mit einem Mitarbeiter ab. Sollte die Überprüfung der Bonität jedoch negativ ausfallen, muss ein Mitarbeiter der Bank den Antrag manuell bearbeiten. Hier bietet BPEL4People Instrumente, entsprechende Interaktionen mit menschlichen Akteuren zu implementieren (vgl. [38]).

4.2.1 BPEL4People Einsatzszenarien

In diesem Paragraph werden die verschiedenen Szenarien beschrieben, in denen die Interaktion mit menschlichen Akteuren notwendig ist, um das Einsatzgebiet von BPEL4People einzugrenzen.

HumanTask

Im Geschäftsprozess wird die interagierende Person als eine Aktivität definiert, wobei die einzelnen Kommunikationsvorgänge als HumanTask bezeichnet werden. Jeder Benutzer erhält eine persönliche TaskList, in der vom System alle Arbeitsschritte angezeigt werden, die vom jeweiligen Benutzer bearbeitet werden müssen. Wird ein Aufgabenschritt durch den Benutzer angenommen, wird zu Beginn des Bearbeitungsschrittes eine Meldung mit den, für die Abarbeitung benötigten Informationen angezeigt [48].

Rollenkonzept

In manchen Situationen darf nicht jeder Mitarbeiter im gleichen Umfang mit dem Prozess interagieren. Als Beispiel sei die Bearbeitung von Reiseanträgen genannt: Jeder Mitarbeiter kann einen Reiseantrag stellen und an Vorgesetzte weiterleiten. Teilzeitmitarbeiter oder studentische Hilfskräfte dürfen einen solchen Antragsprozess jedoch nur in Kooperation mit einem Vorgesetzten durchführen. Hierbei ist es sinnvoll, einen eigenen Geschäftsprozess zu modellieren, der die entsprechenden Personen mit den BPEL-Modellen zu verknüpfen.

Erfassen von zeitlichen Kriterien

Bei der Abarbeitung von verschiedenen Aufgaben kann es dazu kommen, dass ein Prozess auf die Eingabe von Daten wartet, wobei eine gewisse Wartezeit toleriert werden kann. Um Blockierungen an dieser Stelle zu vermeiden, können Timeouts als maximale Wartezeiten definiert werden. Wird eine solche maximale Wartezeit überschritten, löst der Timeout eine Aktion aus. Hier kann der Benutzer auf verschiedene Arten reagieren. Eine Möglichkeit wäre, auf die angeforderten Daten zu verzichten und den Prozess mit dem vorhandenen Datensatz fortzusetzen. Eine Alternative besteht darin, weiter auf die angeforderten Daten zu warten. Eine dritte Möglichkeit ist, die angeforderten Daten manuell einzugeben und den Prozess entsprechend fortzusetzen. Um diese Funktionalität ordnungsgemäß abzubilden, ist ein Prozessverwalter notwendig, der die Entscheidungen an dieser Stelle adäquat treffen kann. Dieser muss über die entsprechenden Qualifikationen verfügen und den Prozess als solchen überschauen können, um die Konsequenzen der Entscheidung, z. B. bei Prozessfortsetzung unter Verzicht auf die Daten absehen zu können [48].

Erweiterte Interaktionsvorlagen

Das **Vier-Augen-Prinzip** ist eine oft genutzte Vorgehensweise in sicherheitskritischen Bereichen, wie z. B. in der Medizin oder im Bankwesen, in denen eine gegenseitige Kontrolle helfen soll, Fehler zu vermeiden. Hierbei müssen Entscheidungen durch zwei voneinander unabhängige Personen getroffen werden. Des Weiteren werden Eskalationsmodelle definiert. Für den Fall, dass eine bestimmte Aufgabe nicht in der vorgesehenen Zeit abgeschlossen

ist, wird eine entsprechende Mitteilung an einen, im Voraus definierten Empfänger versandt. Je nach den technischen Möglichkeiten oder der Dringlichkeit der Nachricht kann aus verschiedenen Übertragungstechniken, wie z. B. eMail, Fax oder SMS gewählt werden. Die weitere Steuerung ist somit an den Empfänger der Nachricht delegiert [48].

4.2.2 Eigenschaften von BPEL4People

Personenintegration

Die Beziehung zwischen Personen und Prozessen kann auf verschiedene Art und Weise abgebildet werden. Der Prozessinitiator ist derjenige, der eine Instanz des Prozesses erzeugt, wohingegen Personen, die an dem Prozess teilnehmen und die Instanz beeinflussen können als Stakeholder bezeichnet werden. Aus Prozesssicht handelt es sich bei den Bearbeitern um so genannte „potential owners“, wobei hier der Status bei Prozssstart vom potentiellen auf den aktuellen Besitzer wechselt. Zu jedem Prozess wird ein Verwalter definiert. Die verschiedenen, am Prozess teilnehmenden Personen und Personengruppen werden dem Prozess mittels Verknüpfungen zugeordnet.

Aufgaben

Zu jeder Aufgabe wird eine Beschreibung erstellt, die dem Benutzer bei der Bearbeitung der Aufgabe helfen soll. Um möglichst übersichtliche Informationen zu bieten wird in der Aufgabenlisten nur eine Zusammenfassung, sowie deren Priorität angezeigt, um einen ersten Eindruck von der Aufgabe zu vermitteln. Erst, wenn der Benutzer die jeweilige Aufgabe zur Bearbeitung ausgewählt hat, wird die detaillierte Beschreibung angezeigt. Abgesehen davon wird jeder Aufgabe eine Frist zugeordnet, um die gesamte Ausführungszeit nicht ausufern zu lassen.

4.3 Transformation nach BPEL

Als Ausführungsumgebung für BPEL ist ein Application Server mit BPEL und BPEL4People Unterstützung vorgesehen. Ausgehend vom erarbeiteten Konzept müssen die beschriebenen Elemente nach BPEL bzw. BPEL4People transformiert werden, so dass es möglich ist den beschriebenen Prozess auf dem Application Server auszuführen. Die Transformation zwischen den Elementen des in Kapitel 3.3 beschriebenen Konzepts, der sogenannten „Domain specific Language (DSL)“ und BPEL erfolgt statisch während der Implementierung des Prozesselements. Prinzipiell wäre eine automatische Transformation auch möglich, die jedoch zunächst nicht betrachtet wird, da eine Vielzahl von Sonderfällen beachtet werden müssen und die hierfür benötigte Zeit nicht zur Verfügung steht. Die Herausforderung bei der Transformation liegt darin, die Semantik der Sprachelemente geeignet in BPEL zu beschreiben, so dass der Funktionsumfang der DSL vollständig und korrekt abgebildet wird.

Ziel bei der Umsetzung in BPEL ist es technische Probleme auf der Ebene der Instruktionen zu lösen, so dass auf den strukturell höheren Ebenen, wie dem Aktionsplan oder dem Workflow, der Fokus in der Umsetzung der Ablauflogik liegt. Konkrete technische Lösungen, wie beispielsweise die Umsetzung eines zentralen Datenspeicher werden nicht näher betrachtet, da dies nicht im Problembereich der PG liegt. Damit im Rahmen von Ad-hoc-Modifikationen die Identifizierung der zurzeit ausgeführten Prozessschritte einfacher ist und ggf. die Betrachtung einzelner Statuswerte differenzierter erfolgen kann, erhält jedes BPEL Element einen im Prozess eindeutigen Identifier. Auf Basis der Beschreibungen in den nachfolgenden Kapitel ist es möglich die einzelnen Elemente der DSL in BPEL zu übersetzen.

4.3.1 Workflows

Ein Workflow beschreibt einen vollständigen Prozess, der auf dem Application Server ausgeführt wird. Wie zuvor beschrieben, kann ein Workflow Entscheider und Aktionspläne enthalten. Darüber hinaus werden auf der Ebene des Workflows Varianten berücksichtigt. In Abbildung 4.2 wird die fachliche Ansicht eines Workflows dargestellt. Die technische

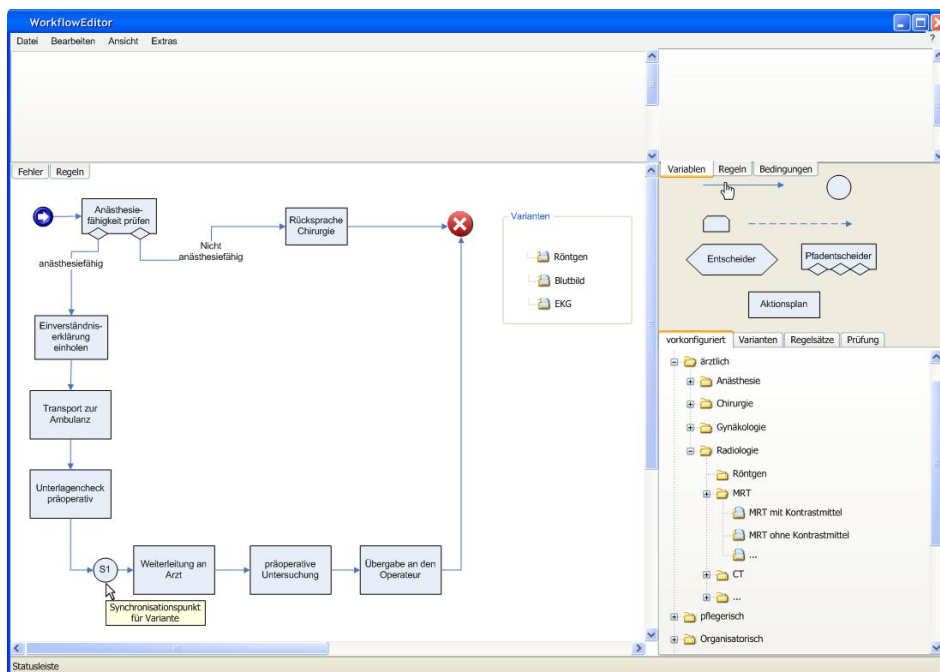


Abbildung 4.2: Fachansicht Prozess .

Umsetzung dieses Beispiels wird in Abbildung 4.3 dargestellt und im folgenden Abschnitt eingehender beschrieben. Die Aufrufe der Aktionspläne werden als Web Service Aufrufe umgesetzt, da die Aktionspläne wiederverwendbare Komponenten sind, die in weiteren Workflows aufgerufen werden. Das Element Entscheider wird ebenfalls über einen Serviceaufruf realisiert. Zunächst war angedacht den Aufruf direkt mit Hilfe von HumanTasks

umzusetzen. Dieser Lösungsansatz wurde jedoch verworfen, da dies bereits eine zu konkrete technische Umsetzung darstellt. Stattdessen wird die benötigte Logik ebenfalls in einen Softwareservice ausgelagert, der als Rückgabe die Entscheidung für einen der möglichen Pfade liefert.

4.3.2 Aktionspläne



Abbildung 4.3: BPEL-Ansicht Aktionsplan

Aktionspläne (vgl. Abbildung 4.3) stellen ein Bündel von Konstrukten dar. Jeder Aktions-

plan wird durch einen separaten BPEL Prozess umgesetzt und durch die Receive-Aktivität ReceiveStartCommand initialisiert. Bei der eingehenden Nachricht handelt es sich um die Quittung über den Abschluss des vorangegangenen Prozesses, welcher die Nachricht aus-sendet. Der Normalfall ist, dass anschließend mit der Bearbeitung des Aktionsplanes begon-nen werden kann. Sollte jedoch im weiteren Verlauf (s. u.) der ABORT-Fall eintreten, so sagt eine globale Variable aus, dass der eigentliche Inhalt des Aktionsplans übergangen wird und die nächste Aktivität direkt SendWorkResult ist und damit den aktuellen Aktionsplan beendet. Somit kann sichergestellt werden, dass kein weiterer Aktionsplan ausgeführt wird, sobald einmal die ABORT-Variable entsprechend belegt wurde.

Regeln

Gemäß der Ausführungssemantik dienen Regeln dazu, sicherzustellen, dass bestimmte Be-dingungen erfüllt sind, bevor Instruktionen oder ganze Aktionspläne begonnen werden. Dies geschieht konkret durch die Anwendung einer Invoke-Operation TryToGetRuleData, die über einen Webservice die erforderlichen Daten ermittelt. Dieser Webservice gibt entweder die angeforderten Daten direkt zurück oder meldet, dass die gewünschte Information (noch) nicht verfügbar ist. Falls die Daten direkt erreichbar sind, wird der Pfad IfRuleDataAccessi-ble durchlaufen und die Assign-Aktivität AssignRuleData ausgeführt. Diese sorgt dafür, dass die empfangenen Daten auch in den Prozess übernommen werden. Anderenfalls wird erst eine gewisse Zeit auf die Daten gewartet (hier 10 Min.), dies wird über die Aktivität Wait-ForIncomingRuleData realisiert. Wird in dieser Zeit eine Nachricht mit den erforderlichen Daten empfangen, werden sie durch das onMessage-Event GetRuleDataOP entsprechend ausgewertet. Läuft die Wartezeit ohne Ereignis ab, reagiert die onAlarm-Aktivität und wird der Nutzer informiert, dass die Daten noch fehlen und anschließend aufgefordert, die Da-ten einzugeben. Je nach Regeltyp ist das entsprechende Verhalten zu modellieren. Hierbei werden die empfangenden Daten entsprechend des Regeltyps in die dafür vorgesehenen Variablen übermittelt. Dies wird mit Hilfe der AssignRuleData-Aktivität durchgeführt. Die Ausführungssemantik wird in der IfReadyForWork-Verzweigung umgesetzt. An dieser Stelle wird auch geAn dieser Stelle werden die in der AssignRuleData-Aktivität belegten Variablen entsprechend ausgewertet.

1. WAIT-Regel

Hierbei wurde bei der Auswertung der Regel festgestellt, dass mit der eigentlichen Arbeit des Aktionsplanes begonnen werden kann, sobald alle notwendigen Daten vor-liegen.

2. SKIP-Regel

Sollte eine SKIP-Regel zu wahr ausgewertet werden, führt dies dazu, dass keine In-struktionen ausgeführt werden, was dem Überspringen des gesamten Aktionsplans entspricht. Der nachfolgende Aktionsplan muss jedoch aktiviert werden. Hier wird als nächstes direkt die SendWorkResult-Aktivität ausgeführt. Ein Fall in dem es zum

Überspringen eines Aktionsplanes kommt, tritt z. B. ein, wenn ein Patient Röntgenbilder vorlegt und keine erstellt werden müssen. Hierbei würden jedoch die erforderlichen Daten bereits im ersten Teil, der Regelprüfung in den Prozess eingebunden, was dazu führt, dass die `SendWorkResult`-Aktivität eine korrekte Ausgabe erzeugt.

3. ABORT-Regel

Ein ABORT führt dazu, dass der eigentliche Inhalt des aktuellen Aktionsplans übergangen wird und zur Belegung einer globalen Variablen, die in jedem Aktionsplan abgefragt wird. Somit ist eine korrekte Weiterbearbeitung seitens externer Prozesse sichergestellt, d.h. jeder weitere Aktionsplan wird noch gestartet der Workflow entsprechend beendet.

Instruktionen

Bei Instruktionen handelt es sich um Arbeitsschritte aus dem Klinikalltag. Die Ausführung der Instruktionen findet statt, falls die `IfReadyForWork`-Verzweigung positiv ausgewertet und der entsprechende Pfad abgearbeitet wird. Wurde in der Regelauswertung festgestellt, dass alle Bedingungen erfüllt wurden, kann mit der Arbeit begonnen werden. Hierbei werden alle Arbeitsschritte (WorkItem 1 und 2 sequentiell nacheinander, Instruktion 3 unabhängig davon) ausgeführt, indem Webservices aufgerufen werden, die die Abarbeitung der klinischen Aktivitäten auslösen.

Abschließend erfolgt der Aufruf der Reply-Aktivität `SendWorkResult`, mit deren Hilfe nachfolgende Aktionspläne über den Abschluss des aktuellen informiert werden.

4.3.3 Ad-hoc-Modifikationen

Ad-hoc-Modifikationen sind auf Ebene der Workflows möglich. Auf dieser Ebene basiert die technische Umsetzung in BPEL auf dem Aufruf von bereits vorhandenen Prozessen. Für Änderungen zur Laufzeit müssen die Einschränkungen, die im Konzept erläutert sind, durch den Editor berücksichtigt werden. Ad-hoc-Modifikationen sind somit, bezogen auf die Transformation in BPEL, problemlos möglich. Einschränkungen können jedoch aufgrund eines Status oder aufgrund von Randbedingungen vorhanden sein. Ist eine Änderung möglich, wird der vollständige BPEL Prozess inklusive einer Änderungshistorie an den Application Server weitergereicht. Auf weitere Details wird in Kapitel 6 eingegangen.

4.3.4 Varianten

Varianten sind häufig durchgeführte Änderungen von Workflows. Eine Variante stellt im Kern somit eine Ad-hoc-Modifikation dar, bei der jedoch schon zum Zeitpunkt des Designs auf ggf. auftretende Abhängigkeiten Rücksicht genommen wurde, so dass für den Fachanwender die Anwendung einer Variante in der Regel einfacher, als die Durchführung einer Ad-hoc-Modifikation ist. Für Varianten gelten somit die gleichen Einschränkungen

und Grundlagen wie für ad hoc Änderungen. Folglich lassen sich Varianten problemlos in BPEL umsetzen. Vom Modell aus ist es vorgesehen, dass für Varianten bekannt ist, an welcher Stelle des Prozesses sie eingefügt werden können. In der Ansicht für Fachanwender wird dies durch spezielle Symbole verdeutlicht. Auf Ebene der technischen Umsetzung ist diese Stelle jedoch nicht weiter zu berücksichtigen. Für den Fall, dass mehrere Varianten parallel (im Sinne von parallelen Ausführungspfaden) eingefügt werden, wird mit Hilfe von AND-JOINS sichergestellt, dass im Kontrollfluss parallel angeordnete Aktionspläne allesamt abgeschlossen sein müssen, bevor mit dem nächsten Schritt im Prozess fortgefahren werden kann.

4.3.5 BPEL Umsetzung eines 1-out-of-M-Join

Im Rahmen der Umsetzung des Workflow-Modells in BPEL wurde festgestellt, dass es keine Möglichkeit gibt, einen **1-out-of-M join** eins zu eins umzusetzen. Bei der Vereinigung von zwei oder mehr Pfaden an einem Synchronisationspunkt muss (in BPEL) von jedem Pfad eine abschließende Meldung gesendet worden sein. Hierbei handelt es sich um Quittungs-Meldungen jedes Pfades, die Aufschluß darüber geben, ob die entsprechende Aktion erfolgreich ausgeführt wurde oder fehlgeschlagen ist. Erst wenn von allen eingehenden Pfaden die Meldungen empfangen wurden, werden auf den Synchronisationspunkt folgende Aktionen gestartet.

Im DynamicFlow-Konzept wird jedoch davon ausgegangen, dass es Situationen geben kann, bei denen der Pfad an einer Stelle in mehrere Verzweigungen aufgespalten wird und die auf einen Zusammenschluß folgenden Aktionen gestartet werden können, sobald ein einziger Vorgänger abgeschlossen wurde.

Ein mögliches Beispiel ist das parallele Abarbeiten der folgenden Aktionen, wie in Abb. 4.4 gezeigt:

- Überprüfen des Pulses
- Überprüfen der Atmung

Hierbei führt der positive Test einer Vitalfunktion direkt zur Weiterbehandlung. Wenn ein Puls gemessen wird oder sobald die Atmung festgestellt wird, liegt kein Herzstillstand vor und der Patient muss weiterbehandelt werden. Das bedeutet, sobald eine Aktion abgeschlossen ist, kann die auf den Zusammenschluss folgende Aktion ausgeführt werden. Sofern z. B. Atmungsaktivität festgestellt wurde, ist es unerheblich, ob und wann die Pulsmessung erfolgt. Die Pulsfrequenz kann zwar weiter gemessen werden, ob die Messung jedoch im Gange oder schon abgeschlossen ist, ist hierbei für die Fortsetzung des Pfades irrelevant. Die Aktionen werden begonnen und irgendwann abgeschlossen, haben jedoch auf den weiteren Verlauf keinen Einfluss.

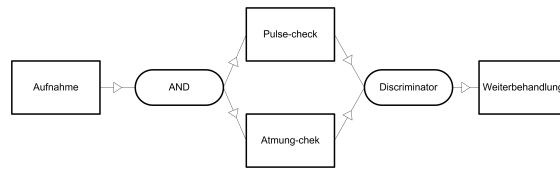


Abbildung 4.4: Behandlung eines Herzstillstandes

Structured Discriminator

Ein mögliches Vorgehen im obigen Fall ist das *Structured Discriminator*-Pattern. Dies ermöglicht das Zusammenführen von zwei oder mehr Pfaden zu einem Folgepfad in der Art, dass der Abschluss einer Aktivität dazu führt, dass die folgende aktiviert wird und Abschlüsse der anderen Aktivitäten vor dem Zusammenschluss weder eine neue Aktivierung noch sonst einen Einfluss auf den folgenden Pfad haben.

Der Ablauf des *Structured Discriminator* wird in Abb. 4.5 dargestellt. Die Notation $()$ an den Transitionen steht für ein einfaches, untypisiertes Signal. In diesem Beispiel befindet sich ein solches Signal in p_2 , was bedeutet, dass der Discriminator aktivierbar ist. Das erste Signal, das an einem der Eingänge i_1 bis i_m empfangen wird, bewirkt, dass der Discriminator aktiviert ist und ein Signal im Ausgang o_1 erzeugt wird. Ein weiteres untypisiertes Signal wird in p_3 erzeugt. Dies bedeutet, dass der Discriminator bereits gesendet hat (und nachfolgende Aktionen gestartet wurden) jedoch noch nicht zurückgesetzt wurde. Später eintreffende Signale haben keinen Einfluss und es werden auch keine neuen Ausgaben erzeugt. Sobald an allen Eingängen Signale empfangen wurden, wird der Discriminator zurückgesetzt und kann erneut aktiviert werden. Dies ist der Fall, wenn $m - 1$ Signale an p_1 eingegangen sind und die Reset-Transition aktiviert werden kann.

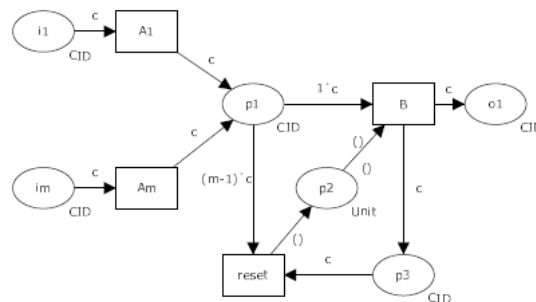


Abbildung 4.5: Structured discriminator pattern (Quelle: [22])

Es existieren zwei Bedingungen bei Anwendung dieses Konzepts:

1. Wenn der *Structured Discriminator* einmal aktiviert und noch nicht zurückgesetzt ist, dann ist kein Empfang eines anderen Signals möglich, weder für den aktivierten Eingang noch für irgendeinen Eingang eines mehrstufigen Signals.

Cancelling discriminator pattern

Die zweite Alternative ist der *Cancelling Discriminator*. Hierbei werden alle weiteren Aktivitäten auf den eingehenden Kanten davor bewahrt, aktiviert zu werden, sobald die erste Aktivität abgeschlossen wird. Stattdessen werden alle übrigen Kanten in einen bypass-mode versetzt, wobei die verbleibenden Aktivitäten übersprungen werden. Dies führt dazu, dass der Discriminator unter Umständen schneller zurückgesetzt werden kann, da nicht alle (möglicherweise überflüssigen) Aktivitäten ausgeführt werden müssen. Die Funktion des *Cancelling Discriminator* ist in Abbildung 4.7 dargestellt.

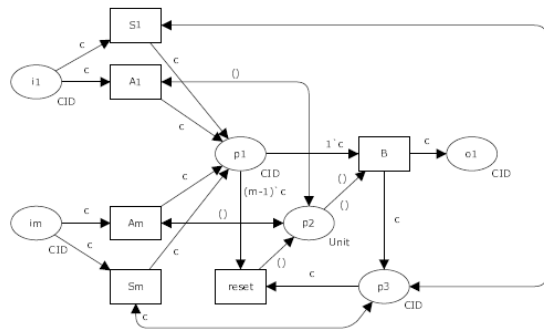


Abbildung 4.7: Cancelling discriminator pattern (Quelle: [22])

Ein mögliches Problem bei Anwendung dieses Konzepts ist, dass ein Empfangsfehler auf einem Signaleingang dazu führen kann, dass der Prozess an dieser Stelle hängen bleibt. Dies kann auch dazu führen, dass folgende Instanzen verzögert werden.

Discriminator-Konzept und BPEL

Leider wird ein solches Discriminator-Konzept in BPEL nicht unterstützt. Eine Lösung durch die Verwendung von `<link>` Konstrukten und eine entsprechende Join-Condition (z. B. `link1==true or link2==true or link3==true`) ist ebenfalls nicht möglich, da die BPEL-Spezifikation vorsieht, dass eine Join-Condition erst ausgewertet wird, wenn der Zustand aller Links bekannt ist.

Zusammenfassung

Nach der Analyse steht fest, dass in der vorliegenden BPEL-Spezifikation (Version 2.0) keine Möglichkeit besteht, die gewünschte Semantik in Standard BPEL-Konstrukten umzusetzen. Da jedoch im aktuellen Projektstand keine Anwendungsfälle aufgetreten sind, die ein oben beschriebenes Konzept erfordern, wird auf die weitere Suche von Lösungen zu diesem Problem zunächst verzichtet. Die zu untersuchenden Szenarien lassen sich unter Verwendung der Standard-BPEL-Konstrukte realisieren.

5 Produktauswahl

Sowohl die Modellierung der Prozesse als auch deren Ausführung sollte auf geeigneten Plattformen stattfinden. In den folgenden beiden Unterkapiteln werden zunächst Kriterienkataloge für verschiedene Modellierungsumgebungen (Editoren) und Ausführungsumgebungen (Engines) aufgestellt. Abgesehen davon werden die verfügbaren Applikationen anhand der Kriterien bewertet und ihr Einsatzpotential für die PG abgeleitet.

5.1 Editorauswahl

Zunächst werden verschiedene Editoren bewertet, wobei sowohl die Möglichkeit bestand ein Produkt lediglich zu verwenden als auch Anpassungen daran vorzunehmen. Es wurde jeweils untersucht, ob die Anwendungen kostenlos, quelloffen oder kommerziell zu beziehen sind. Anschließend werden ähnliche Kriterien auf die verschiedenen BPEL-Engines angewendet, welche ebenfalls auf die Verfügbarkeit (kostenlos, quelloffen oder kommerziell) und ihre Anpassungsfähigkeit für Zwecke der PG hin untersucht wurden.

5.1.1 Oracle BPEL Designer

Der Oracle BPEL Process Manager [16] bietet eine auf dem BPEL-Standard basierende Rahmenkonstruktion, um die Prozesse zu entwerfen, einzusetzen, zu überwachen und zu verwalten. Abgesehen davon werden folgende Eigenschaften unterstützt [103]:

- Web Service Standards, wie XML, SOAP und WSDL
- Korrelation zwischen asynchronen Nachrichten
- Service Orientierte Architektur (SOA)
- Parallele Behandlung der Aufgaben
- Fehlerbehandlung und Ausnahmenverwaltung während der Entwurfszeit und der Laufzeit
- Ereignis- und Zeitüberwachung, sowie -Notifikationen
- Kompensationsmechanismus für die Implementierung länger laufender Prozesse
- Anpassbarkeit und Zuverlässigkeit der Prozesse

- Versionskontrolle
- Installation auf verschiedenen Betriebssystemen und Integration mehrerer Applikationsserver und Datenbanken

Oracle BPEL Process Manager unterstützt zwei verschiedene BPEL-Entwurfs-Umgebungen, zum einen ist dies der JDeveloper BPEL Designer, zum anderen der Eclipse BPEL Designer. Der JDeveloper BPEL Designer wird im Oracle JDeveloper 10g integriert. Oracle JDeveloper 10g ist eine integrierte Umgebung, um die Applikationen und Web Services durch Java, XML und SQL-Standard aufzubauen. Eclipse BPEL Designer ist als Plug-In in der Eclipse 3.0 Plattform integriert und benutzt Standard-BPEL, d. h. die entworfenen Prozessen können mit anderen BPEL-Servern benutzt werden [103].

5.1.2 ActiveBPEL Designer

Die Firma Active Endpoints [24] bietet einen BPEL Designer und eine BPEL Engine an. Zu Beginn der Projektgruppe fand ein Versionswechsel statt. In der angekündigten Version werden einige der gestellten Anforderungen erfüllt. Besonders herauszustellen ist, dass es der einzige BPEL Designer am Markt ist, der zum aktuellen Zeitpunkt die Möglichkeit bietet mit BPEL4People Elementen zu modellieren und BPEL4People Code erzeugt. Das Tool stellt eine Ansicht zur Modellierung zur Verfügung, in der ein Prozess mit Hilfe von grafischen BPEL Elementen abgebildet wird. Technisch basiert der BPEL Designer auf der Eclipse Plattform und dem Eclipse BPEL Plug-In und stellt damit gute Möglichkeiten für ggf. notwendige Erweiterungen zur Verfügung. Die Erweiterungen, die durch Active VOS an dem Eclipse BPEL Plug-In vorgenommen wurden, stehen aktuell nicht als Open Source Version zur Verfügung. Eine solche Version ist bereits seit Monaten angekündigt, jedoch aktuell nicht verfügbar. Insgesamt hinterlässt das Tool einen sehr guten Eindruck. Insbesondere bei einer Verwendung der Active VOS BPEL Engine zur Ausführung des erzeugten BPEL Codes bietet das Tool eine gute Unterstützung beim Debugging und Deployment.

5.1.3 Eclipse BPEL Designer

Von der Open Source Community Eclipse wird ein eigener BPEL-Designer [4] und ein BPEL Quelltext Editor angeboten. Das Projekt ist als Eclipse Plug-In implementiert und kann mithilfe der Eclipse Umgebung problemlos installiert werden.

Es stehen zwei unterschiedliche Sichten zur Verfügung:

- BPEL-Code Editor - um Anpassungen im BPEL-Quelltext vorzunehmen
- BPEL Modellierer - um BPEL-Prozesse per „drag-and-drop“ zu modellieren.

Das Plug-In verfügt über eine Komponentenpalette, in der alle BPEL-Konstrukte gruppiert sind.

Das Plug-In verfügt über die folgenden Funktionalitäten:

- Visuelle Modellierung eines BPEL-Prozesses
- Darstellung des BPEL-Prozesses als EMF-Modell
- Möglichkeit die Prozesse zu deployen und auszuführen
- Debugging

Das BPEL Plug-In ist ein Open Source Projekt und somit ist der Quelltext online verfügbar, was es erlaubt das Projekt weiter zu entwickeln. Im Rahmen der Projektgruppe ist es gelungen, einige neue Konstrukte ins Modell und in den Designer erfolgreich zu integrieren.

Das Projekt wird von der Eclipse-Community weiterentwickelt, und wird unter anderem von grossen Software-Unternehmen (Oracle, IBM, Intel) unterstützt.

5.1.4 Eclipse BPMN Modeler

Das BPMN Modeler Projekt wird von der Open Source Community Eclipse [5] entwickelt und angeboten. Das Projekt ist als Eclipse Plug-In implementiert und kann über die Eclipse Webseite oder über die Updatefunktion in der Programmierumgebung Eclipse installiert werden.

Das Projekt umfasst die Spezifikationen BPMN 1.0 und BPMN 1.1. Das Plug-In benutzt unter Anderem auch andere Eclipse Projekte und Frameworks, z. B. für grafische Aspekte wird GMF und für semantisches Modell EMF benutzt.

Die folgenden Aspekte sind beim BPMN Modeler berücksichtigt:

- Es können BPMN Diagramme genutzt werden, um die Workflows zu modellieren.
- Aus dem Diagramm können EMF Objekte generiert werden, um diese in andere Modelle (z.B. BPEL) zu transformieren.
- Die grafischen Elemente können mit Annotationen versehen werden.
- Das Plug-In ist erweiterbar und somit können neue BPMN-Elemente oder andere Konstrukte integriert werden.

Es gibt mit dem aktuellen Projekt jedoch ein Problem, das die Transformation nach BPEL betrifft. Es können in der aktuellen Version vom BPMN Modeler keine BPEL Dateien generiert werden. Um aus den BPMN Dateien BPEL Code zu erzeugen muss jedoch ein Zwischenschritt ausgeführt werden:

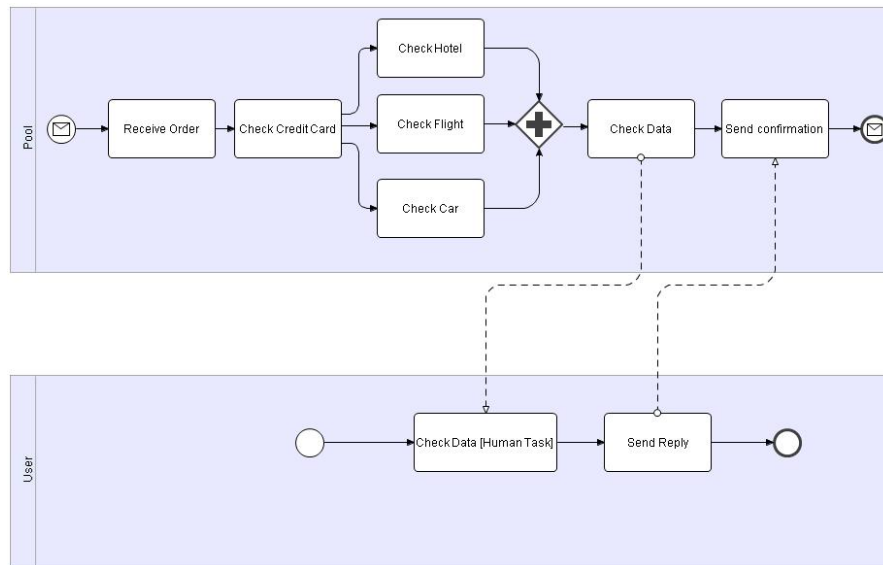


Abbildung 5.1: Die Modellierungsschicht des BPMN Modelers.

- BPMN Diagramme werden mithilfe des Eclipse BPMN Modeler erstellt.
- Die BPMN Diagramme werden mit BABEL Tools [2] in eine Zwischensprache konvertiert.
- Die Dateien in der Zwischensprache werden mithilfe des BPMN2BPEL Tools [2] nach BPEL übersetzt.

Die Tatsache, dass man einen Zwischenschritt benötigt, um den BPEL Code zu erzeugen, spielte eine entscheidende Rolle bei der Ablehnung des BPMN-Modelers und BPMN als Modellierungssprache.

5.1.5 Intalio BPMN Designer

Die Firma Intalio [69] stellt einen eigenen BPMN Editor (Intalio|Designer) [13] zur Verfügung. Wie der Name schon sagt, wird dieser Editor zur Modellierung mithilfe von BPMN benutzt. Die Version von Intalio basiert auf der Eclipse Architektur. Die Modellierungsumgebung verfügt über alle Werkzeuge und Mittel, um problemlos in der BPMN-Sprache zu modellieren. Die Komposition von grafischen Werkzeugen im Intalio BPMN-Designer ist sehr gut strukturiert und benutzerfreundlich. Die erstellten BPMN-Projekte lassen sich auch mit einer eigenen Version von HumanTasks verbinden, indem die sog. XForms genutzt werden. XForms ist eine XML-basierte Sprache für die Entwicklung und die Bearbeitung von Benutzeroberflächen im Internet. Die Modellierung von XForms wird durch eine eigene Sicht unterstützt. Die Modellierungsumgebung verfügt über eine integrierte Palette

mit BPMN-Elementen, die mit Hilfe von „drag-and-drop“ in Diagramme eingebunden werden können. Mithilfe des Intalio|Designer können, im Rahmen einer Entwicklungsumgebung Geschäftsprozesse modelliert, an andere Systeme angebunden und auf von Intalio zur Verfügung gestellte Server hochgeladen werden. Das Intalio|Designer Projekt ist mit anderen Projekten, Intalio|Server [14] und Intalio|Workflow [15] verbunden. Intalio|Server ist ein auf J2EE basierender BPEL 2.0 konformer Server. Intalio|Workflow stellt eine BPEL4People Engine dar. Die im Projekt vorhandenen XForms werden durch AJAX-Technologie unterstützt, wobei die Erstellung von AJAX-Code automatisch durch die Engine erfolgt. Das Projekt ist kein Open Source-Projekt und somit ist keine eigene Weiterentwicklung möglich.

5.1.6 AgilPro

AgilPro [1] ist eine Anwendung zur Modellierung von Prozessen auf der Eclipse Plattform und wurde von der Universität Augsburg [20] mit Partnern aus der freien Wirtschaft entwickelt. Als eine der wenigen Entwicklungsumgebungen stehen dem Benutzer zwei verschiedene Sichten auf den Prozess zur Verfügung. In der fachlichen Ansicht werden die wesentlichen Elemente des Prozesses dargestellt. In der technischen Ansicht erhält der Anwender zusätzliche Möglichkeiten zur Modellierung, so dass der Prozess vervollständigt werden kann. Zur Modellierung verwendet AgilPro ein eigenes Modell, das nach BPEL transformiert wird. Als Ausführungsplattform ist jedoch eine eigene Engine vorgesehen, mit Hilfe derer beispielsweise auch Worddokumente auf Arbeitsstationen zur Verfügung gestellt werden können, so dass diese Dokumente direkt mit in den Prozess integriert sind.

5.1.7 Fazit

Die technische Grundlage für die Umsetzung des Editors zur Erstellung von Workflows wird das Framework Eclipse sein. Für dieses Framework existieren bereits eine Reihe von Erweiterungen die ggf. direkt verwendet werden können, so dass kein Aufwand für Umsetzung bereits verfügbarer Funktionen betrieben werden muss. Auf Basis von Eclipse wird eine Anwendung entwickelt mit deren Hilfe die Prozesse modelliert, umgesetzt und geändert werden können. Dies schließt eine Änderung zur Laufzeit, die sogenannte Ad-hoc-Modifikation mit ein. Ausgehend von den zuvor ermittelten Anforderungen an die verschiedenen Benutzergruppen ist es sinnvoll für jede Benutzergruppe jeweils eine Perspektive zu erstellen. Es wird zwischen der Benutzergruppe der Anwender und der Gruppe der Entwickler unterschieden. Ziel ist es, die Erweiterungen als Plug-In für Eclipse zur Verfügung zu stellen. Der Benutzer kann über die Konfiguration festlegen, welche Perspektive er verwenden möchte. In den nachfolgenden Unterkapiteln werden die Funktionen der einzelnen Perspektiven dargestellt. Bevor jedoch auf den Aufbau bzw. den Inhalt der Perspektiven eingegangen wird, werden die einzelnen Module, die in den Perspektiven verwendet werden, vorgestellt. Auf Details zur technischen Umsetzung wird an dieser Stelle ebenfalls eingegangen.

5.2 Engineauswahl

In der ersten Phase der PG wurden Kriterien für die Auswahl einer geeigneten Engine zur Ausführung der modellierten Workflows aufgestellt. Die aufgestellten Anforderungen sind in Tabelle 5.1 dargestellt.

Da sich die PG zur Umsetzung der Workflows für die Ausführungssprache BPEL entschieden hat, ist die Unterstützung von BPEL durch die Engine natürlich ein wichtiges Kriterium. Theoretisch wäre es dabei nicht von Bedeutung, ob nur der ältere Standard BPEL4WS 1.1 oder auch WS-BPEL 2.0 unterstützt wird. Eine weitere Anforderung ist die Unterstützung menschlicher Aktivitäten. Die PG hat sich hierbei für den Einsatz von BPEL4People entschieden, welches als Erweiterung von WS-BPEL 2.0 spezifiziert ist. Daher wird auch von der Engine eine Unterstützung von BPEL 2.0 gefordert.

Eine weitere wichtige Anforderung war das Vorhandensein von Quelltext, da schon zu Beginn der PG klar war, dass die Umsetzung von Ad-hoc-Modifikationen nicht ohne Eingriffe in die Engine möglich sein wird. Damit scheidet kommerzielle Lösungen, von denen der Quelltext nicht verfügbar ist, direkt aus.

Anforderungen an eine Workflow-Engine	ActiveBPEL	Intalio BPMS
BPEL 2.0	erfüllt	erfüllt, mit Einschränkungen
Abbildung menschlicher Aktivitäten	erfüllt	erfüllt
BPEL4People (nach Spezifikation 1.0)	erfüllt	nicht erfüllt
Open Source (Erweiterbarkeit)	erfüllt, GPL	erfüllt, mit Einschränkungen
Community	erfüllt	erfüllt
Prozessversionierung	erfüllt	erfüllt
Persistenzmechanismus	einrichtbar	einrichtbar

Tabelle 5.1: Anforderungen an eine Ausführungseingine

Im Anschluss an die Festlegung der Kriterien wurden existierende Produkte anhand dieser Kriterien auf die Eignung für den Einsatz innerhalb des Projektes bewertet. Die beiden Kriterien „Open Source“ und „Abbildung menschlicher Aktivitäten“ führten schnell zum Ausschluss der meisten Engines, so dass nur zwei mögliche Alternativen genauer betrachtet wurden: ActiveBPEL Engine und Intalio BPMS. Im folgenden werden beide Engines kurz vorgestellt. Auf die Architektur von ActiveBPEL wird im Kapitel 6.1 detaillierter eingegangen.

5.2.1 Intalio BPMS Server

Die Firma Intalio [69] vertreibt ihren sogenannten BPMS Server in verschiedenen Formen:

- Intalio|BPMS Enterprise Edition
Die kostenpflichtige Enterprise Edition des Intalio BPMS Servers wird ausschließlich

in Binärform vertrieben, und bietet den großen Vorteil eines professionellen, kommerziellen Supports.

- **Intalio|BPMS Community Edition**
Genau wie die kostenpflichtige Enterprise Edition wird die kostenlose Community Edition des BPMS Servers ausschließlich in Binärform vertrieben, bietet allerdings keinen kommerziellen Support. Weiterhin lässt sich diese Version, im Unterschied zur kommerziellen Variante, ausschließlich mit einer MySQL-Datenbank und nicht mit einer beliebigen Datenbank, verbinden. Der Einsatz einer Datenbank ist notwendig, sofern BPEL-Prozesse persistent gespeichert werden sollen. Die Community Edition kann frei für Evaluation, Entwicklung und im Produktivbetrieb eingesetzt werden.
- **Open Source Variante**
Die Open Source Variante des BPMS Servers liegt im Quelltext vor und enthält laut Intalio [93] 95% des Quelltexts der Community und Enterprise Edition. Diese Version muss vom Benutzer selbständig kompiliert und eingerichtet werden. Wie bei der Community Edition ist für diese Version kein professioneller Support verfügbar.

Wie bereits weiter oben beschrieben wäre im Rahmen der PG einzig ein Einsatz der Open Source Variante möglich, da nur hier der Quelltext vorliegt, der für Änderungen an der Engine benötigt wird. Allerdings ist der Einsatz der Open Source Variante leider auch nur bedingt sinnvoll, da diese Variante nur etwa 95% des Codes der anderen Versionen beinhaltet. Es ist auch nicht möglich, sich direkt den benötigten Code herunterzuladen. Vielmehr besteht die Open Source Variante aus den beiden Teilprojekten *Apache ODE* [80] und *Intalio Tempo* [70]. Dabei stellt ODE die eigentliche BPEL-Engine dar, während im Intalio Tempo Projekt weitere Komponenten, die zur Ausführung von Workflows benötigt werden, zusammengefasst werden. Zusätzlich wird noch ein Applikationsserver (Apache Geronimo J2EE) und eine Datenbank (MySQL) benötigt. Die Integrationsleistung dieser Produkte in ein lauffähiges Produkt, die Intalio in der Community Edition und kommerziellen Variante erbracht hat, ist beim Einsatz der Open Source Version vom Entwickler selbst zu erbringen.

5.2.2 ActiveBPEL

Die ActiveBPEL Engine der Firma Active Endpoints [24] wird sowohl in einer kommerziellen Variante, als auch in einer Open Source Variante unter GPL vertrieben. Da der Umfang der Open Source Variante zu einem großen Teil der kommerziellen Variante entspricht, und alle von der PG gestellten Anforderungen erfüllt, ist ein Einsatz dieser Version möglich. Positiv an ActiveBPEL ist die vollständige Umsetzung der Standards WS-BPEL 2.0 und BPEL4People 1.0 (an deren Spezifikation die Firma Active Endpoints aktiv beteiligt war). Ein weiterer Vorteil ist das Vorhandensein eines Web Service, mit dem sich unter anderem neue Prozesse deployen und Statusabfragen durchführen lassen. Da auch dieser Web Service

unter GPL steht, ist der komplette Quelltext vorhanden und kann von der PG unter anderem als Basis für die Umsetzung des erforderlichen Prozessmonitoring verwendet werden (Für weitere Informationen über Interaktionsmöglichkeiten mit der Engine siehe Kapitel 6.4).

5.2.3 Fazit

Die PG hat sich aufgrund der oben genannten Kriterien für den Einsatz der ActiveBPEL Engine entschieden. Problematisch im Vergleich zu Intalio ist höchstens die Tatsache, dass die Weiterentwicklung der Engine alleine in der Hand eines kommerziellen Unternehmens liegt, und nicht wie bei Apache ODE von einer Community vorangetrieben wird. Es hat sich gezeigt, dass die Entscheidung für ActiveBPEL kein Fehler gewesen ist, da der Quelltext einen verständlichen, gut dokumentierten Eindruck macht, so dass zukünftige Erweiterungen und Veränderungen relativ unproblematisch zu realisieren sein sollten. Des Weiteren gab es während des Verlaufs der PG bereits zwei neue Veröffentlichungen des Engine Quelltexts, was für eine aktive Weiterentwicklung seitens Active Endpoints spricht.

Ein weiterer Grund, der gegen einen Einsatz der Engine von Intalio gesprochen hat, ist die Art der Umsetzung von menschlichen Aktivitäten: Im BPEL4People Whitepaper [68] werden fünf verschiedene Möglichkeiten beschrieben, in der endgültigen Spezifikation [67] sind davon nur vier übrig geblieben. Intalio hat sich zur Umsetzung der fünften Variante, in der die BPEL-Engine nicht verändert werden muss, entschieden. Hierbei werden menschliche Aktivitäten als spezieller Web Service dargestellt, der durch normale <invoke>-Aufrufe angestoßen wird. Problematisch dabei ist, dass die Modellierung dadurch aufwändig und unübersichtlich wird. Abschließend sei bemerkt, dass sich der Entscheidungsprozess zwischen den beiden Produkten als relativ langwierig darstellte, und die Wahl von Intalio wohl auch keine unüberwindbaren Probleme mit sich gebracht hätte.

6 Umsetzung

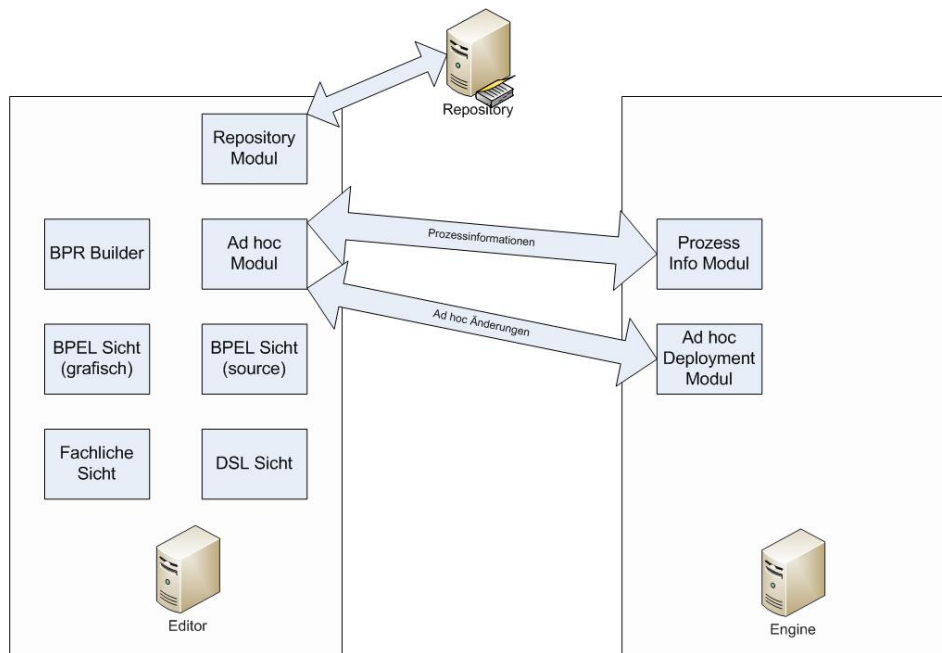


Abbildung 6.1: Architektur

Ausgehend von der Auswahl einer Engine und eines Editors als Grundlage von Erweiterungen ergibt sich der in Abbildung 6.1 dargestellte Aufbau. In der Abbildung sind die beteiligten Systeme und Komponenten einer Umgebung dargestellt, die zur Modellierung, Ausführung und Ad-hoc-Modifikation von Workflows verwendet werden. Im Editor wird der Prozess auf mehreren Detailebenen modelliert und umgesetzt, so dass die verschiedenen Anwendergruppen eine geeignete Sicht auf den Prozess zur Verfügung gestellt bekommen. Da für den vollen Funktionsumfang des Editors noch weitere Systeme angesprochen werden müssen, wird die dafür benötigte Schnittstellenlogik in einzelne Module ausgelagert, so dass die Entwicklung zwischen mehreren Mitgliedern des Teams aufgeteilt werden kann. Die Kommunikation der Module mit ihren Zielsystemen erfolgt über Web Services. Der vollständig implementierte Prozess wird mit Hilfe des BPR-Builders als Datei exportiert. Der BPR-BUILDER wird in Kapitel 6.1.3 (S. Seite 117) detaillierter vorgestellt. Diese Datei enthält alle verwendeten Prozesse, also insbesondere die verwendeten Aktionspläne und Instruktionen, so dass sichergestellt wird, dass der Engine alle benötigten Prozesse zur Verfügung stehen. Diese Datei wird anschließend manuell auf dem Application Server deployed. Wird der BPR-BUILDER direkt durch das Ad-hoc-Modul angesprochen, enthält das Deploymentarchiv

zusätzlich die Änderungshistorie des Editors über die vorgenommenen Ad-hoc-Modifikationen. Das Ad-hoc-Modul ermöglicht die Kommunikation zum Application Server zur Laufzeit des Application Servers. Dazu stellt das Ad-hoc-Modul Anfragen an das Prozess Info Modul des Application Servers. Über diese Schnittstelle kann nach Prozessen gesucht werden, Prozesse können heruntergeladen werden und Informationen zu den Prozessen können abgefragt werden. Dabei kann beispielsweise nach bestimmten Prozessen (beispielsweise nach der Prozessart, dem Name des Patienten oder dem Status des Prozesses) gesucht werden. Nach dem eine Prozessinstanz ausgewählt wurde kann zu dieser Instanz der aktuelle Status abgefragt werden. Das Modul fragt dazu den kompletten Prozess (Deploymentdatei) ab und erhält zusätzlich den aktuellen Status der Instanz. Soll die Prozessinstanz ad hoc geändert werden, wird die Ausführung der betreffenden Instanz auf dem Application Server angehalten und die Statuswerte werden aktualisiert. Während der Durchführung von Änderungen werden diese durch das Modul protokolliert. Durch die zu Anfang erwähnte Zuweisung von eindeutigen Bezeichnern für jedes BPEL Element, kann für jedes Element identifiziert werden welche Änderungsoperation (löschen, hinzufügen, überspringen) vorgenommen wurde. Nach Durchführung der Änderung erstellt das Modul, mit Hilfe des BPR-Builders, eine neue Deploymentdatei und stellt diese dem Application Server über das Ad-hoc-Modul zur Verfügung. Das Repository Modul stellt die Anbindung an das zentrale Verzeichnis zur Ablage der Workflows, der Aktionspläne und Instruktionen zur Verfügung. Mit Hilfe des Moduls kann innerhalb des Repositories nach Inhalten gesucht werden. Aus der Liste der Suchergebnisse können die gewünschten Komponenten per „drag-and-drop“ in der Bearbeitungsansicht abgelegt werden. Des Weiteren sorgt das Modul dafür, dass neue Prozesse im Repository abgelegt werden können.

6.1 Engine

Die PG hat sich in Kapitel 5.2.3 für ActiveBPEL als BPEL Ausführungengine entschieden. Für diese Entscheidung wurden die wesentlichen Eigenschaften der beiden Engines abgewägt.

Für das Verständnis der Umsetzung des PG-Konzeptes wird nun die gewählte Engine ActiveBPEL detaillierter erläutert und dabei wird auf einige technische Details, die grundlegende Architektur und den wesentlichen Umgang mit der ActiveBPEL Engine eingegangen.

6.1.1 Wesentlicher Umgang mit ActiveBPEL

Die Engine benötigt einen Servlet-Container. Von der PG wird Apache Tomcat verwendet und das entsprechende JAVA JDK 5. Als eine gute Kombination hat sich Tomcat 5.5.x und JDK 1.5.x erwiesen. Der Einfachheit halber wird bei erwähnten URLs davon ausgegangen, dass ActiveBPEL auf einem lokalen Applikations-Server läuft und den Port 8080 nutzt. Tomcat wird also mit „localhost:8080“ angesprochen.

Es sind zwei Webseiten verfügbar, die Informationen über die Engine anzeigen, die im Folgenden kurz beschrieben werden.

Axis Web Services Listing

Apache Axis2 [82] ist eine Web Service Plattform, die in ActiveBPEL benutzt wird. Auf der Seite <http://localhost:8080/active-bpel/services> werden alle verfügbaren Web Services angezeigt, da diese von Axis erstellt wird. Obwohl es sich dabei um eine in ActiveBPEL integrierte, angepasste Version von Axis2 handelt, ist die Liste unabhängig von ActiveBPEL. Die Engine gibt jedoch Web Services zum deployen an Axis. Die Liste zeigt verfügbare Methoden in den Web Services und auch ihre WSDL Datei an.

Die Kommunikation mit der Engine ist in großen Teilen über Web Services möglich, die ebenfalls an Axis zum „deployen“ übergeben werden und daher auch in der Liste auftauchen. Auf diese wird weiter unten eingegangen.

ActiveBPEL Engine Administration

Die Seite <http://localhost:8080/BpelAdmin/> stellt administrative Funktionen zum Anzeigen und Bearbeiten von Konfigurationsparametern, deployte Prozesse (siehe Abbildung 6.2) und aktive Prozesse zur Verfügung, die direkt ActiveBPEL betreffen. Die Liste der aktiven Prozesse kann abhängig vom Status, z. B. „completed“ gefiltert werden.

Darüber hinaus ist eine „Receive Queue“ und eine „Alarm Queue“ Ansicht verfügbar. Erstere listet aktive „receive“ und „onMessage“ Aktivitäten auf, also alle Aktivitäten, die gerade in irgendeiner Weise auf eingehende Nachrichten warten. Die „Alarm Queue“ tut dies analog für „onAlarm“ Aktivitäten aller laufenden Prozessinstanzen, also solche Aktivitäten die einen Alarm ausgelöst haben.

Für eine Prozessinstanz kann man die „Process Details Page“ aufrufen. Diese zeigt eine grafische Ansicht oder eine BPEL Code Ansicht der Instanz an. Man kann sich über die Prozess- und Aktivitätseigenschaften und Werte, den Ausführungsstatus jeder Aktivität und die aktuellen Werte von Variablen, Activity Links, Partner Links, Correlation Sets, Fault Compensation und Eventhandler informieren. Insbesondere kann hier ein beliebiger Prozess manuell durch die Anweisung „suspend“ in den Zustand „suspended“, mit der Anweisung „resume“ kann der Zustand „suspended“ wieder in den Zustand „running“ und mit der Anweisung „terminate“ in den Zustand „terminated“ versetzt werden.

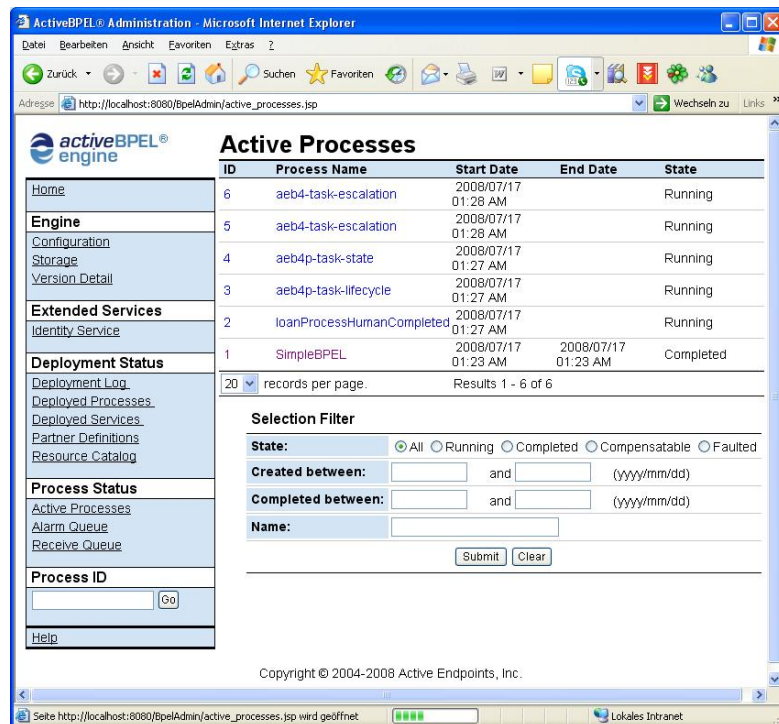


Abbildung 6.2: Active Processes Queue

Die einzelnen Prozesszustände lauten

- „Completed“ → der Prozess wurde abgeschlossen
- „Faulted“ → tritt ein, wenn ein Fehler während der Ausführung geworfen wurde oder der Prozess terminiert wurde
- „Running“ → der Prozess läuft noch
- „Compensatable“ → ein laufender Unterprozess kann zurückgenommen werden
- „Suspended“ → der Prozess wurde angehalten, seine Ausführung kann durch „resume“ wieder fortgesetzt werden.

Die einzelnen Aktivitätszustände lauten

- „Executing“ → die Aktivität wird gerade ausgeführt
- „Ready to Execute“ → die Aktivität ist ausführungsbereit
- „Finished“ → die Aktivität wurde beendet
- „Faulted“ → tritt ein, wenn ein Fehler während der Ausführung geworfen wurde

- „Terminated“ -> tritt ein, wenn der Prozess manuell terminiert wurde
- „Dead Path“ -> die Aktivität liegt in einem Pfad des Prozesses, der nicht mehr erreicht werden kann
- „Suspended“ -> die Aktivität wurde angehalten
- „Inactive“ (Initialzustand einer Aktivität) oder terminiert

Man beachte, dass ein Zustand „Active“, den man durchaus erwarten könnte, durch die Zustände „Executing“ und „Ready to Execute“ differenziert wird. Abbildung 6.3 zeigt die interne Aktivitätszustände und deren Übergänge von ActiveBPEL. Man beachte, dass diese Zustände nicht denen aus dem in der PG entwickelten Konzept entsprechen.

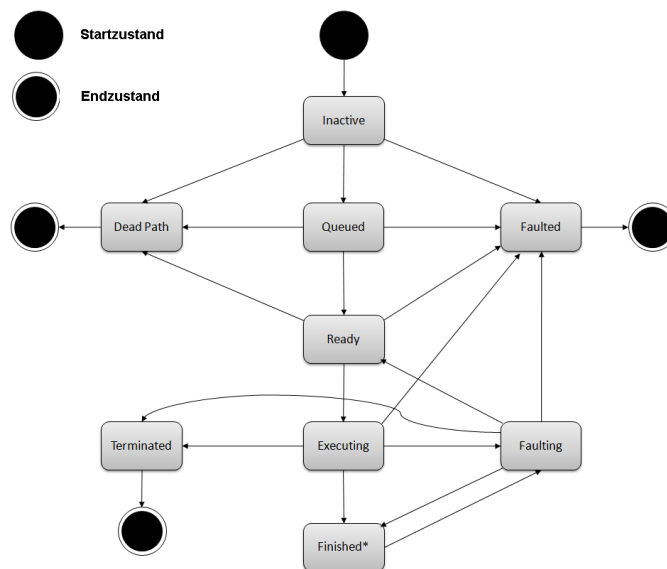


Abbildung 6.3: Zustände in ActiveBPEL .

Task Inbox

Wenn in einem Prozess eine Interaktion mit einem Menschen nötig ist, wird dafür ein „Task“ erzeugt und an die entsprechende „User Inbox“ geschickt, die zu der benötigten Rolle bzw. der benötigten Gruppe gehören, die in dem „Identity Service“ gepflegt sind. Alle Tasks werden auch immer an die „Inbox“-en der Administratoren geschickt, weil diese die „Tasks“ bestimmten Usern zuweisen können, wie das Abbildung 6.4 veranschaulicht. Ein User kann einen „Task“ beanspruchen („claim“), was bedeutet, dass er diesen Task bearbeiten möchte und der Task dadurch keinen anderen Usern mehr zur Bearbeitung

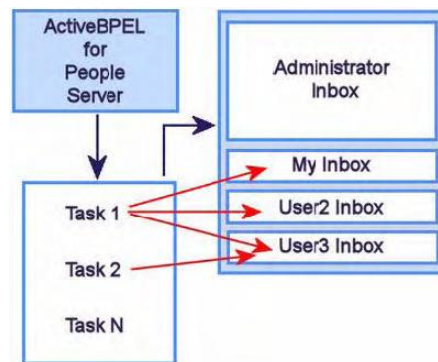


Abbildung 6.4: Zuweisung von Tasks an Inboxes verschiedener qualifizierter User [24].

zur Verfügung steht, solange der User diesen nicht wieder freigibt („release“). Die „Inbox“ wird über die URL <http://localhost:8080/aetask> aufgerufen. Die Abbildungen 8.1 und 8.2 zeigen eine modifizierte Version der „Inbox“ und die entsprechende Bearbeitungsansicht der Aufgaben. In dem Kapitel 8.2 wird auch weiter auf eine mögliche Anpassung der „Inbox“ eingegangen. Hier soll nun noch etwas zur Technik der „Inbox“ erwähnt werden. Die „Inbox“ ist eine Web-Applikation, die für einen „Human Task“ per XSL ein rendering durchführt. Insbesondere können vorhandene Daten natürlich auch durch XSLT transformiert werden. Das Standardrendering kann modifiziert werden und beim Deployment eines Prozesses als XSL Datei mitgeliefert werden. Darüber hinaus kann die „Inbox“ teilweise auch durch eigene Webseiten ersetzt werden.

Identitätsverwaltung

Zur Identitäts- und Rollenverwaltung kann die Engine an einen LDAP Verzeichnisdienst angeschlossen werden oder alternativ entsprechende Einträge in der Datei `tomcat-users.xml`, die im Verzeichnis `/CATALINA_HOME/conf/` liegt, ergänzt werden. Welche Quelle für die Identitätsinformationen genutzt werden soll, teilt man ActiveBPEL auf den Administration Pages mit. Insbesondere bei der Untersuchung der „Inbox“ hat die PG von der XML-Datei Gebrauch gemacht. Diese Datei lässt sich auch recht komfortabel über das „Admin Pack“ [81] für Tomcat pflegen.

Persistenz

Standardmäßig läuft die Engine lediglich im „In-Memory“ Modus, bei dem sämtliche Prozessinstanzen beim Beenden der Engine verloren gehen. Man kann jedoch einen Persistenzmodus aktivieren, der die Zustände in einer Datenbank sichert. Dafür sind im Wesentlichen folgende Voraussetzungen nötig.

- Datenbanksystem
Die PG verwendet dafür Derby.
- JDBC Treiber
Entsprechend der Datenbank verwendet die PG den JDBC Treiber „org.apache.derby.jdbc.EmbeddedDriver“
- aeEngine database ddl
Das bei der Engine mitgelieferte Datenbankskript, um die notwendigen Tabellen und Schlüssel zu erzeugen.
- Tomcat Datasource Setup
Die Datenquelle der Tomcat-Umgebung bekannt machen.
- „aeEngineConfig.xml“ anpassen
Mit dieser Datei kann man die Engine konfigurieren und aktiviert letztlich auch den Persistenz Modus von ActiveBPEL.

6.1.2 Deployment eines BPEL-Prozesses

Um einen Prozess in der Engine zum Deployment, muss dieser in einem BPR-Archiv vorliegen, das neben der BPEL Datei auch andere benötigte Artefakte des Prozesses, wie z. B. WSDL Dateien enthält. Wird dieses Archiv dem Verzeichnis /CATALINA_HOME/bpr/ hinzugefügt, wird die Änderung durch die Hot-Deployment Fähigkeit der Engine sofort registriert und das Deployment für diesen Prozess wird ausgeführt. Nach Abschluss dieses Vorgangs kann der Prozess direkt verwendet werden.

In der Engine wird aus dem BPEL-Prozess ein Prozess des Typs AeProcessDef erstellt. Für jede BPEL Aktivität gibt es eine Spezialisierung der Klasse AeActivityDef (AeActivitySequenceDef, AeActivityWaitDef, AeActivityFlowDef, AeActivityEmptyDef, ...). Nachdem alle benötigten Def-Objekte angelegt wurden, wird der BPEL-Prozess einer statischen Validierung unterzogen, bei der der BPEL-Prozess z. B. auf ungenutzte Variablen, nicht definierte Variablen oder Links geprüft wird. Ist dies alles erfolgreich abgeschlossen ist der Prozess deployt und wartet auf eine eingehende Nachricht.

Trifft diese ein wird in der Regel eine neue Instanz des Prozesses erzeugt. Dabei gibt es für jedes AeActivityDef-Objekt eine Spezialisierung der Klasse AeAbstractBpelObject (AeActivitySequenceImpl, AeActivityWaitImpl, AeActivityFlowImpl, AeActivityEmptyImpl, ...). Im Gegensatz zu Def-Objekten haben Objekte vom Typ AeAbstractBpelObject einen Zustand (Inactive, Ready, Executing, Dead Path, ...). Mögliche Zustandsübergänge veranschaulicht die Abbildung 6.5.

Die Engine führt zu jedem Zeitpunkt immer nur genau eine Aktivität/Objekt des Prozesses aus. In einem Flow werden daher alle Kindaktivitäten in die Ausführungsqueue eingereiht und der Flow ist dann beendet, wenn alle Kinder beendet sind. Es wird also immer nur eine Aktivität aus der Queue ausgeführt. Die „Parallelität“ eines BPEL Flows wird folgendermaßen erreicht:

Die Engine hat verschiedene Manager, die sich um bestimmte Arbeiten kümmern. Die Implementierung einer „WAIT“-Aktivität z. B. würde nach Ablauf einer Deadline vom Alarm Manager benachrichtigt und dadurch beendet. Dadurch werden Arbeiten auf andere Module ausgelagert und die Engine kann mit der Abarbeitung des restlichen Prozesses fortfahren, ohne auf die Beendigung dieser Arbeiten warten zu müssen. Die einzelnen Module des Managerkonzeptes werden in 6.1.4 vorgestellt. Weitere Details zu den für das Deployment benötigten Artefakten werden in 6.1.3 gegeben.

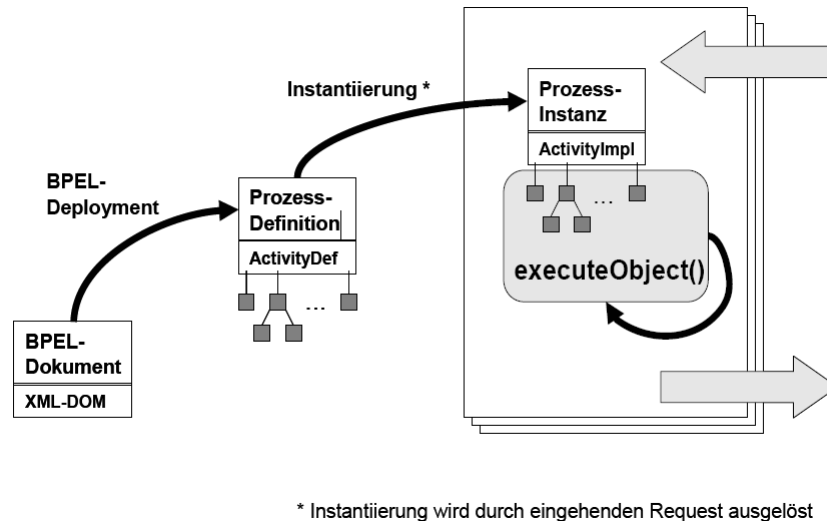


Abbildung 6.5: Prozesslebenszyklus [79]

Um das BPR-Archiv zu erzeugen, müssen die benötigten Artefakte zuerst in einer Struktur einzelner Ordner abgelegt werden, anschließend werden die von ActiveBPEL erforderten Informationen erzeugt, und schließlich daraus das BPR-Archiv.

Z. B. haben wir den Prozess `my_prozes.bpel` und zwei zugehörige WSDL-Dateien `service1.wsdl` und `service2.wsdl`. In diesem Beispiel wird ein Verzeichnis namens `mybpel` erzeugt und darin in Unterverzeichnissen die für den Prozess benötigten Dateien abgelegt. Die Unterverzeichnisse der Verzeichnisstruktur lauten

- `bpel`
- `deploy`
- `META-INF`
- `wsdl`
- `partners (optional)`

Das Verzeichnis partners ist nicht nötig, es sei denn, Partnerdefinitions Dateien (.pdef) sind vorhanden. So besitzt das BPR-Archiv von my_process.bpel die Struktur in Abbildung 6.6.

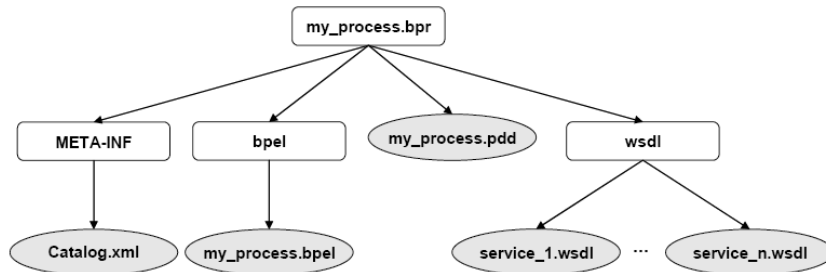


Abbildung 6.6: Das BPR-Archiv von my_process.bpel

Anschließend wird das BPR-Archiv in den Servlet-Container kopiert. Bei Apache Tomcat bedeutet dies das Kopieren des BPR-Archivs in /CATALINA_HOME/bpr/. Wenn ActiveBPEL bereits läuft, wird das BPR-Archiv erkannt und eingelesen, da die Engine periodisch das Deploymentverzeichnis nach Änderungen scannt (Hot Deployment). Anschließend wird der oben beschriebene Deploymentprozess durchgeführt und die Definitionsobjekte erzeugt. Dann ist die BPEL-Prozessinstanz in der BPEL-Engine bereits verfügbar.

6.1.3 Deploymentarchiv

Inhalt des BPR-Archivs

Das BPR-Archiv umfasst, wie oben bereits erwähnt, folgende Struktur von Teilverzeichnissen:

- bpel: enthält BPEL-Datei (.bpel)
- deploy: enthält Process Deployment Descriptor Datei (.pdd)
- META-INF: enthält Catalog-Datei (Catalog.xml)
- wsdl: enthält WSDL-Dateien (.wsdl)
- partners (optional): enthält Partner Definition Datei (.pdef)

Process Deployment Descriptor (.pdd)

Für jeden zu deployenden BPEL-Prozess ist eine eigene Process Deployment Descriptor Datei (.pdd) nötig. Diese XML-Datei liefert der Engine die benötigten Informationen über den BPEL-Prozess. Das Element <process> enthält Partnerlinks und WSDL-Referenzen.

Das Schema für .pdd Datei ist in <http://schemas.active-endpoints.com/pdd/2006/08/pdd.xsd> zu finden. Listing 6.1 zeigt die Struktur einer pdd-Datei.

Listing 6.1: Struktur einer pdd-Datei (gekürzt)

```
<process name="qname" location="relativeDeploymentLocation">
  <partnerLinks>
    <partnerLink name="ncname">
      <partnerRole
        endpointReference="static|dynamic|invoker|principal">
        [... endpoint reference ...]?
      </partnerRole>?
      <myRole service="name" allowedRoles="namelist"?
        binding="MSG|RPC"/>?
    </partnerLink>+
  </partnerLinks>
  <wsdlReferences>
    <wsdl namespace="uri" location="uri"/>+
  </wsdlReferences>?
</process>
```

Dabei enthält das Attribut „relativeDeploymentLocation“ den relativen Pfad zur zugehörigen BPEL-Datei.

Partnerlinks beschreiben mögliche Rollen von Partnern im Prozess.

Während des Deployments wird auch die Binding-Strategie bzw. die Kriterien festgelegt, mit welcher die genaue „Endpoint Reference“ der bis dahin noch abstrakten Partnerlinks bestimmt wird. Die ActiveBPEL Engine benutzt dafür eine der Methoden, die durch folgende Parameter in der pdd-Datei festgelegt werden können:

- **Dynamic**
Der Parameter „dynamic“ besagt, dass die Endpoint Reference dynamisch innerhalb des BPEL-Prozesses zur Verfügung gestellt wird. Sie wird anhand einer copy-Operation einer <assign>-Aktivität dem entsprechenden partnerLink zugeordnet.
- **Invoker**
Die Parameter „invoker“ fordert, dass es sich bei dem Aufruf um einen asynchronen Prozessaufruf handelt, d. h. der Prozess antwortet nicht mit einem <reply> sondern mittels einer <invoke>-Aktivität. Eine Endpoint Reference wird aus dem Header einer instanzierenden SOAP-Nachricht entnommen und dem Partnerlink der <invoke>-Aktivität zugeordnet, welche den „call back“ ausführt.
- **Principal**
Ähnlich wie bei der Parameter „invoker“ wird hier die Endpoint Reference für den Partnerlink an den der „call back“ eines asynchronen Prozessaufrufs geschickt wird, zum Instanzierungszeitpunkt festgelegt. Allerdings wird hier durch die instanzierende Nachricht nicht die Endpoint Reference direkt übergeben, sondern lediglich eine

Kennung des Nutzers des Prozesses. Anhand dieser Kennung wird dann mittels eines Lookups in der Engine die zu verwendende Endpoint Reference aus der vorher deployten Partner Definition (.pdef Datei) ermittelt.

- **Static**
Der Parameter „static“ bewirkt, dass die Endpoint Reference bei der Instanziierung aus dem Process Deployment Descriptor ausgelesen und dem Partnerlink zugeordnet wird.

Der Partnerlink für eine „Knie-Operation“ BPEL-Prozess sieht z. B. wie in Listing 6.2 aus.

Listing 6.2: Beispiel von partnerlinks

```
<partnerLinks
  xmlns:Ins="http://medizinOperationen.org/wsd/knieOperation">
  <!-- Partner Link fuer KnieOperation-Anfrage vom Patienten -->
  <partnerLink name="Patiente" type="Ins:knieOPLinkType">
    <myRole service="knieOP" allowedRoles="" binding="RPC" />
  </partnerLink>
  <!-- Partner Link fuer KnieOperation-Bewilligung vom Arzt -->
  <partnerLink name="Arzt" type="Ins:knieOPLinkType">
    <partnerRole endpointReference="static">
      <wsa:EndpointReference
        xmlns:approver="http://tempuri.org/services/knieOpArzt">
        <wsa:Address>Arzt:anyURI</wsa:Address>
        <wsa:ServiceName
          portName="SOAPPort">Arzt:knieOpArzt</wsa:ServiceName>
        </wsa:EndpointReference>
      </partnerRole>
    </partnerLink>
  </partnerLinks>
```

Das Element „<wsdlReferences>“ listet alle WSDL-Dateien bezüglich eines BPEL-Prozesses auf. Diese werden von der Engine benutzt, um die Repräsentationen der WSDL im Speicher zu erzeugen. Listing 6.3 zeigt ein Beispiel für „<wsdlReferences>“-Elemente.

Listing 6.3: Ein mögliches Element <wsdlReferences>

```
<wsdlReferences>
  <wsdl
    namespace="http://tempuri.org/services/knieOpDefinitionen"
    location="wsdl/knieOPDefinitionen.wsdl"/>
  <wsdl
    namespace="http://medizinOperationen.org/wsd/knieOpDurchfuehrung"
    location="wsdl/knieOpDurchfuehrung.wsdl"/>
</wsdlReferences>
```

Catalog.xml

Die Catalog-Datei liefert der Engine Information um die WSDL-Dateien innerhalb eines BPR-Archivs zu finden. Außerdem enthält sie mögliche Information über XML-Schema Dateien in Form von XSD-Dateien. Sie befindet sich im META-INF Verzeichnis.

Listing 6.4: Beispiel einer Catalog.xml

```
<?xml version="1.0">
<catalog
  xmlns="http://schemas.active-endpoints.com/catalog/2006/07/catalog.xsd">
  <wsdlEntry location="project://knieOperation.wsdl"
    classpath="/wsdlknieOperation.wsdl" />
  <wsdlEntry location="project://knieOpDefinition.wsdl"
    classpath="/wsdlknieOpDefinition.wsdl" />
  <schemaEntry location="projekt:/schema/medizinOperation.xsd"
    classpath="schema/medizinOperation.xsd" />
</catalog>
```

Das Attribut „location“ bestimmt die Zuordnung zu einer WSDL-Datei durch eine der folgenden Methoden. Es ist definiert durch

- entweder das Attribut „location“ eines „<wsdl>“ Elementes innerhalb der „<wsdlReferenzen>“ Sektion einer .pdd-Datei
- oder das Attribut „location“ eines „<import>“ Elementes in einer WSDL-Datei.

Partner Definition (.pdef)

Listing 6.5: Beispiel für ein .pdef

```
<partnerDefinition principal="name">
  <partnerLinkType name="qname">
    <role name="ncname" authtype="basic" username="xx"
      password="yy">
      ... endpoint reference ...
    </role>*
  </partnerLinkType>*
</partnerDefinition>
```

Partnerlinks beschreiben die Beziehung zwischen Partnern. Partner Definition Dateien sind nicht für alle BPEL-Prozesse nötig, sondern nur für solche, die die oben beschriebene Principal-Binding Strategie nutzen. Diese Datei wird benutzt, um die benötigten Informationen für eine Authentifizierung liefern.

BPR-Builder

Um das Deployment von Prozessen zu automatisieren wurde eine entsprechende Funktion, der BPR-Builder implementiert. Der BPR-Builder soll ein entsprechendes BPR-Archiv aus einer vom Editor erzeugten BPEL-Datei und ihren zugehörigen WSDL-Dateien erstellen. Das erzeugte BPR-Archiv wird in das entsprechende Verzeichnis der Engine kopiert, woraufhin diese mit dem Deployment beginnt. Der BPR-Builder wurde als eine Funktion im Editor integriert und stellt eine Interaktion zwischen den Editor und der Engine dar. Die Abbildung 6.7 beschreibt die Struktur des BPR-Builders. Die Pfeile-Richtung zeigt die Aufrufrichtung (von aufrufenden Objekten nach aufgerufenen Objekte).

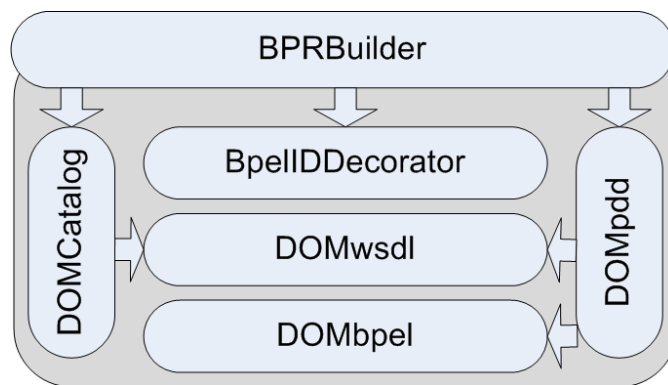


Abbildung 6.7: BPRBuilder

Der folgende Abschnitt beschreibt die Funktionsweise des BPR-Builder. Der Aufruf des BPR-Builder im Editor erfolgt mit dem Verzeichnis, das die BPEL-Datei enthält als Parameter. Somit kann der BPR-Builder die BPEL-Datei und die zugehörigen WSDL-Dateien bestimmen. Zuerst wird das Verzeichnis „META-INF“ für Catalog-Datei (Catalog.xml) erzeugt. Dabei wird der Inhalt von Catalog.xml durch Auslesen der WSDL-Dateien ermittelt, was mit Hilfe von Objekten der Klasse DOMwsdl ermöglicht wird. Anschließend wird die Datei des „Process Deployment Descriptor“ (.pdd Datei) erstellt. In diesem Schritt werden Objekte der Klassen DOMbpel und DOMwsdl genutzt, um die BPEL-Datei und die dazugehörigen WSDL-Dateien zu verarbeiten und entsprechende Informationen zu gewinnen. Um spätere Modifikation von laufenden Prozessinstanzen zu ermöglichen (vgl. Kapitel 6.5), muss eine Möglichkeit zum Vergleich von Prozessinstanzen eines BPEL-Prozesses vorgegeben werden. Dieses ist Aufgabe der Klasse BPELIDDecorator. Dabei wird ein Attribut zur Identifizierung jedes Elements in der BPEL-Datei hinzugefügt. Dieses Attribut erlaubt einen späteren Inhaltsvergleich von Prozessinstanzen eines BPEL-Prozesses. Nachdem die BPEL-Datei dekoriert wird, wird sie an Stelle der originalen ersetzt. Das Listing 6.6 zeigt eine noch nicht behandelte BPEL-Datei.

Listing 6.6: Eine BPEL-Datei vor Dekorierung

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

```
<bpel:process
  xmlns:abx="http://www.activebpel.org/bpel/extension"
  xmlns:b4p="http://www.example.org/BPEL4People"
  ...
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Workflow"
  suppressJoinFailure="yes" targetNamespace="http://example">
  ...
  <bpel:flow>
    <bpel:links>
      <bpel:link name="L1"/>
      <bpel:link name="L2"/>
      <bpel:link name="L3"/>
    </bpel:links>
  ...
  <bpel:assign name="Assign666toResp">
    ...
    <bpel:copy>
      <bpel:from>666</bpel:from>
      <bpel:to variable="NewOperationResponse"/>
    </bpel:copy>
  </bpel:assign>
</bpel:flow>
</bpel:process>
```

Listing 6.7 zeigt eine dekorierte BPEL-Datei, hier werden neben einer eindeutigen ID für jedes Element noch zusätzliche Attribute und Namensräume (sogenannte „Namespaces“) hinzugefügt, die spätere Ad-hoc-Modifikationen ermöglicht.

Listing 6.7: Eine BPEL-Datei nach Dekorierung

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel=
  "http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:abx="http://www.activebpel.org/bpel/extension"
  xmlns:b4p="http://www.example.org/BPEL4People"
  ...
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:nsdf="DfEngineExtensions" name="Workflow"
  suppressJoinFailure="yes" targetNamespace="http://example"
  nsdf:maxId="1053" nsdf:adHocEnabled="true">
<bpel:extensions nsdf:id="1001">
  <bpel:extension mustUnderstand="no"
    namespace="DfEngineExtensions" nsdf:id="1002" />
</bpel:extensions>
...
  <bpel:flow nsdf:id="1011">
  <bpel:links nsdf:id="1012">
```

```

    <bpel:link name="L1" nsdf:id="1013" />
    <bpel:link name="L2" nsdf:id="1014" />
    <bpel:link name="L3" nsdf:id="1015" />
  </bpel:links>
  ...
  <bpel:assign name="Assign666toResp" nsdf:id="1048">
    ...
    <bpel:copy nsdf:id="1051">
      <bpel:from nsdf:id="1052">666</bpel:from>
      <bpel:to variable="NewOperationResponse" nsdf:id="1053" />
    </bpel:copy>
  </bpel:assign>
</bpel:flow>
</bpel:process>

```

Falls ein neues Element hinzugefügt wird, bekommt es den nächsthöchsten ID-Wert. Zur Veranschaulichung wird in Listing 6.8 zuerst der Link „L2“ mit dem nsdf:id=„1014“ gelöst und ein neuer Link „L4“ hinzugefügt. Der höchsten vergebene ID-Wert ist nsdf:maxId=„1053“. Nach dem Entfernen von Link „L2“ wird der ID-Wert „1014“ freigesetzt. Trotzdem wird dieser nun unbesetzte ID-Wert „1014“ nicht für den neu hinzuzufügenden Link „L4“ genommen. In der dekorierten BPEL-Datei bekommt der Link „L4“ stattdessen den nächsthöchsten ID-Wert, nämlich „1054“. Anhand diesen einmal gegebenen ID-Werte kann jeder Inhaltsvergleich von Prozessdefinitionen durchgeführt werden, was eine Identifizierung der Unterschiede ermöglicht. Dieses dient den späteren Modifikationsvorgang.

Listing 6.8: Eine modifizierte BPEL-Datei nach Dekorierung

```

<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel=
  "http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:abx="http://www.activebpel.org/bpel/extension"
  xmlns:b4p="http://www.example.org/BPEL4People"
  ...
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:nsdf="DfEngineExtensions" name="Workflow"
  suppressJoinFailure="yes" targetNamespace="http://example"
  nsdf:maxId="1054" nsdf:adHocEnabled="true">
<bpel:extensions nsdf:id="1001">
  <bpel:extension mustUnderstand="no"
    namespace="DfEngineExtensions" nsdf:id="1002" />
</bpel:extensions>
  ...
  <bpel:flow nsdf:id="1011">
  <bpel:links nsdf:id="1012">
    <bpel:link name="L1" nsdf:id="1013" />
    <bpel:link name="L3" nsdf:id="1015" />
    <bpel:link name="L4" nsdf:id="1054" />
  </bpel:links>
  </bpel:flow>
</bpel:process>

```

```
</bpel:links>
...
<bpel:assign name="Assign666toResp" nsdf:id="1048">
  ...
  <bpel:copy nsdf:id="1051">
    <bpel:from nsdf:id="1052">666</bpel:from>
    <bpel:to variable="NewOperationResponse" nsdf:id="1053" />
  </bpel:copy>
</bpel:assign>
</bpel:flow>
</bpel:process>
```

Anschließend wird das BPR-Archiv erzeugt, was einem jar-Archiv aus dieser Ordnerstruktur entspricht. Als letzter Schritt wird das BPR-Archiv in den Servlet-Container kopiert wird, infolgedessen die Engine mit dem Deployment und der Ausführung des neuen BPEL-Prozesses beginnt.

6.1.4 Architektur ActiveBPEL

Es wurde bereits ein Teil der ActiveBPEL Architektur angesprochen. Im Folgenden wird das Bild nun vervollständigt. Einen groben Überblick gibt die Abbildung 6.8. ActiveBPEL ist vollständig in Java implementiert. Wie bereits erwähnt, läuft ActiveBPEL in einem Servlet-Container und nutzt als Web Service Plattform AXIS2. Ein Event-Framework stellt Funktionalitäten bereit, um z. B. eigene Event-Listener zu realisieren. Kern der Architektur ist die Business Process Engine, die z. B. über das Partner Adressing für dynamische URNs ihre Endpoint References festlegt. Eine solche dynamische Prozessdefinition kann über diese Zuordnung in fremde Systemlandschaften genutzt werden. Die Engine sorgt für die Prozess-Erzeugung und deren Management und hält dafür entsprechend dem Deployment zugehörige Prozessdefinitionen und Instanzen. Verwaltbar ist die Engine über die Engine Administrator Schnittstelle. Ein weiterer grundlegender Aspekt ist das Auslagern einiger wichtiger Funtionalitäten auf sogenannte Manager. [24]

- Process Manager
Erzeugt Prozessinstanzen und verwaltet das Sichern und Wiederherstellen ihrer Prozess-Statistiken
- Queue Manager
Verwaltet die Receive Queues und Correlations
- Alarm Manager
Verwaltet Timer für „on Alarms“ und „waits“
- Cluster Manager (nur bei der kommerziellen Version)
Verwaltet allgemeine Beziehungen zwischen verschiedenen Engines und verwaltet daraus resultierende Fehlschläge

- Storage Manager
Kümmert sich um die Interaktion mit der angeschlossenen Datenbank
- Task Manager
Verwaltet Human Tasks

Darüber hinaus bietet die Engine die meisten Funktionen der „Engine Administration Pages“ auch als Web Service an.

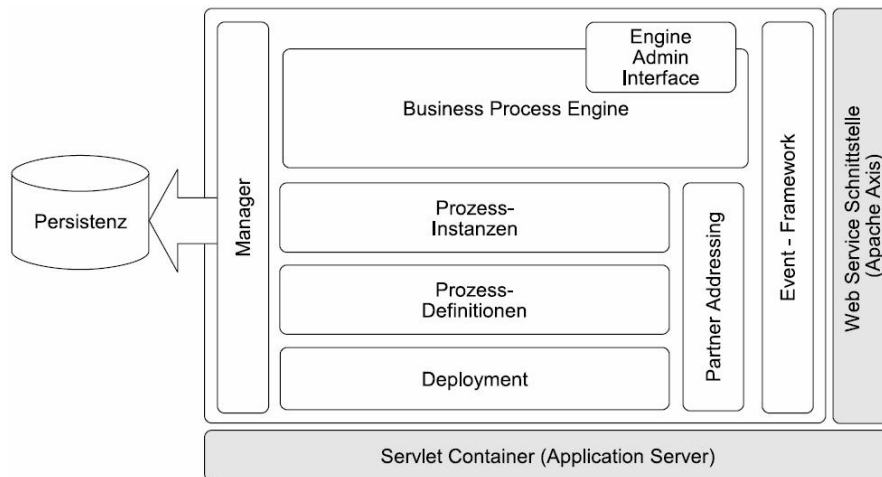


Abbildung 6.8: Grundlegende Architektur von ActiveBPEL . [105, S. 36]

6.2 Modellierung durch Designer

Für die Zwecke der PG wurde ein eigener Designer auf der Basis von bestehenden Open-Source Projekten, wie z. B. das BPEL-Plug-In für Eclipse, entwickelt. Das Eclipse-Framework bietet viele Erweiterungsmöglichkeiten und lässt sich auch an die Ziele der PG anpassen. Die drei Hauptbausteine des Designers sind *BPEL-Editor*, *BPEL-Repository* und *Eclipse-EMF Framework*.

Der Designer kann sowohl für den Entwurf und die Entwicklung von BPEL- bzw. XML-Projekten genutzt werden, als auch, speziell für PG-Zwecke, für den Entwurf von Aktionsplänen bzw. die Modellierung der Abläufe im Klinikalltag.

Es wurden einige von der PG selbst entwickelte Bausteine für die BPEL-Modellierung benutzt: wie z. B. *CommentPlaceholder* oder *HumanTask*.

Der Designer hilft dem Entwickler dabei, die BPEL-Projekte in eine deploybare Form zu bringen, um die BPEL-Prozesse in der BPEL-Engine ausführen zu lassen. Es besteht auch die Möglichkeit die aktuell eingespielten Prozesse nach ihrem *Prozesszustand* zu sortieren und die Prozessdateien von dem Server auf den Entwicklungsrechner zu laden.

6.2.1 Perspektiven

Die Anwendung zur Entwicklung und Bearbeitung von Workflows basiert auf dem Eclipse-Framework. In Eclipse ist es üblich verschiedene fachliche Aspekte über geeignete Sichten, in Form von Perspektiven zur Verfügung zu stellen. Zur Erledigung der anfallenden Tätigkeiten wird eine Anwendung mit zwei Perspektiven zur Verfügung gestellt. Die Perspektive „Modellierung“ dient der Modellierung von Workflows auf den unterschiedlichen Abstraktionsstufen und eine weitere Perspektive „Ad-hoc-Perspektive“ dient zur Durchführung von Ad-hoc-Modifikationen. Eine Perspektive besteht dabei aus mehreren Modulen, die in den verschiedenen Perspektiven wiederverwendet werden können.

6.2.1.1 Editor

Der Editor unterstützt Ärzte bei der Arbeit mit den Aktionsplänen. Die Arbeit mit dem Editor kann von unterschiedlichen Sichtweisen erfolgen: man kann in der BPEL-Sicht den resultierenden BPEL-Code betrachten, in der fachlichen Sicht werden die einzelne Aktionspläne erstellt.

Fachliche Sicht

Die fachliche Sicht wird von den Ärzten benutzt, um die Workflows mit Hilfe von Aktionsplänen zu erstellen. In dieser Sicht steht dem Arzt eine Palette von Aktionsplänen zur Verfügung. Mithilfe der Palette werden die Prozesse auf fachlicher Ebene modelliert und entwickelt. Die Implementierung erfolgt hierbei durch einfaches „drag-and-drop“.

Die Palette mit den Komponenten enthält auch ein Repository, auf das zur Laufzeit zugegriffen werden kann. Außerdem enthält die Palette folgende Bausteine:

- Aktionsplan - die Vorlage eines Aktionsplanes.
- Manueller Entscheider - die Vorlage eines Entscheiders (Aufforderung zu einer konkreten Angabe).
- Automatischer Entscheider - die Vorlage für einen automatischen Entscheider (Auswahl eines Pfades aus mehreren Pfaden).

Abgesehen davon besteht die Möglichkeit, nach den laufenden Prozessen zu suchen. Jeder Prozess wird in einer Liste mit mehreren Parametern, wie *Prozessname* oder auch *Prozesszustand* dargestellt (siehe Abbildung 6.9).

Des Weiteren kann die Liste von Prozessen gezielt nach bestimmten Parametern bzw. Variablenwerten gefiltert werden (vorausgesetzt, dass die Variablen in der Prozessdefinition vorhanden sind).

Prozessname	PID	Prozesszustand
Workflow_01e39b51-1815-487b-8659-0958922f52ed	710	COMPLETED
Workflow_0a217f46-608f-433f-904a-9806515741e0	709	COMPLETED
Workflow_d1ba0092-0547-44ae-8b73-0c2fc4835403	708	COMPLETED
Workflow_8935e6d1-cdab-473b-a982-d247037a7316	707	COMPLETED
Workflow_1c82e319-64b7-46cb-b2a1-bc71b02e1673	706	COMPLETED
Workflow_38e0b745-952d-452a-b18a-348303f570b7	705	COMPLETED
Workflow_7d710f0f-5204-4a81-8162-c798f04abc52	704	COMPLETED
Workflow_5942f8bc-7343-4950-ab53-bf3e90916ccf	703	COMPLETED
Workflow_4d2352b3-8bba-4d1c-acab-21af4beed1af	702	COMPLETED
Workflow_6e41cd8c-7cb7-4296-a4f9-d1d12c9cec05	701	COMPLETED
Workflow_7b23a267-58da-4959-8a85-4f3d0a69d28b	601	SUSPENDED
Workflow_5bb3ed78-0e82-4fbb-88d6-4ed93db99644	502	COMPLETED
Workflow_bbf0f00e-8e28-4b05-90e4-763629d1f4f3	501	COMPLETED
Workflow_37571e74-6732-4c02-a50c-64122ac8c18f	401	COMPLETED

Patientenname eingeben:

Prozesse filtern

Abbildung 6.9: Liste von auf dem Server installierten Prozessen

Ad-hoc-Sicht

Diese Sicht dient zur Kommunikation zwischen dem Designer und der Engine, die den aktuellen Prozess ausführt. In der Sicht wird zuerst eine Instanz des aktuellen Prozesses ausgewählt und die Information über diese Instanz von der Engine angefordert.

Wie man sieht, werden in der Sicht verschiedene Informationsarten dargestellt:

- Protokoll - Angaben zu jedem Aktionsplan, der schon ausgeführt wurde.
- Prozessdiagramm - der aktuelle Prozessinstanz mit den farbigen Aktionsplänen.

Die Aktionspläne können folgende Farben haben:

- Grün - Aktionsplan wurde ohne Fehler ausgeführt.
- Gelb - Aktionsplan ist aktiv und befindet sich in der Ausführung.
- Rot - bei der Ausführung des Aktionsplans sind Fehler aufgetreten.

Es besteht die Möglichkeit aus mehreren vordefinierten Varianten zu wählen und die ausgewählte Elemente ins Diagramm hinzuzufügen. In dem Beispiel [6.11](#) werden zwei Varianten

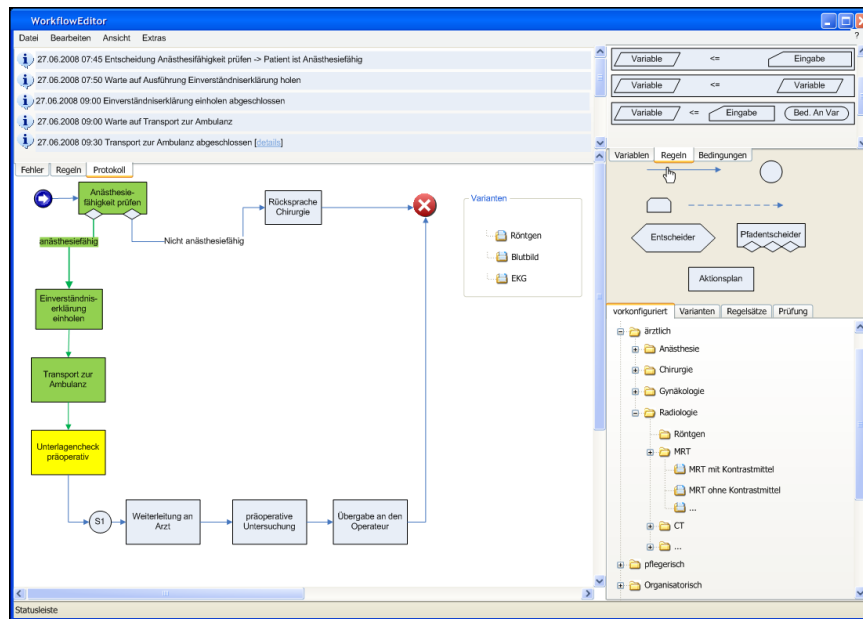


Abbildung 6.10: Designstudie: Die Darstellung einer Prozessinstanz in der „Ad-hoc-Sicht“.

„Blutbild“ und „Röntgen“ parallel nach dem Aktionsplan „Unterlagencheck präoperativ“ hinzugefügt und über einen Synchronisator mit den restlichen Aktionsplänen verbunden.

Die Synchronisation erfolgt durch Warten an dem entsprechenden Synchronisator-Elementen. Sobald alle eingehenden Kanten aktiviert wurden, wird durch den Synchronisator der nächste Aktionsplan aktiviert.

DSL Sicht

In der DSL-Sicht werden die bereits beschriebenen Aktionspläne editiert. Jeden Aktionsplan kann man mit Doppelklick zum Editieren öffnen.

Wie bereits erklärt, besteht ein Aktionsplan aus mehreren Sektionen, die editierbar sein können.

Für jede Sektion gibt es eine entsprechende Sektion in der Palette, wo alle Elemente platziert sind, die zur Sektion gehören. So gibt es z. B. die Sektion „Regeln“, wo die Regelvorlagen zu finden sind. Hier ein Beispiel zu Regeln:

Die DSL-Sicht unterscheidet sich von der fachlichen Sicht dadurch, dass man in der DSL-Sicht auch die einzelnen Instruktionen eines Aktionsplanes betrachten kann.

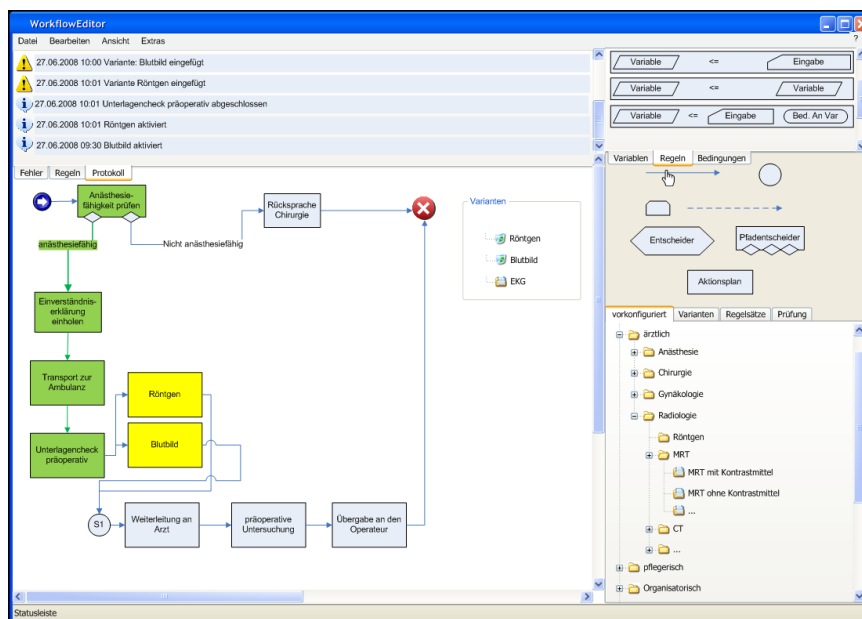


Abbildung 6.11: Designstudie: Ad-hoc-Modifikation: Zwei Varianten werden hinzugefügt.

BPEL Sicht

Die BPEL-Sicht dient dazu, BPEL-Prozesse visuell zu modellieren. Die Sicht besteht aus mehreren Bereichen:

- Palette - alle BPEL-Elemente nach Kategorien gruppiert (s. Abbildung 6.15 auf Seite 133).
- Diagrammbereich - ein BPEL-Prozess grafisch dargestellt (s. Abbildung 6.16 auf Seite 134).
- Eigenschaften - die Eigenschaften der ausgewählten BPEL-Elementen.
- Prozess - Variablen und Partnerlinks vom aktuellen Prozess.

Die Konstruktion eines BPEL-Prozesses erfolgt per „drag-and-drop“, die fertigen Elemente können aus der Palette entnommen und daraus ein Diagramm konstruiert werden.

Die BPEL-Elemente besitzen Eigenschaften, die man in den entsprechenden Views auch editieren bzw. anschauen kann. Alle Änderungen sind sofort in der Quelltextansicht zu sehen (s. Abbildung 6.17 auf Seite 134). Die Dateien lassen sich in Projekte gruppieren, sodass sich z.B. die WSDL- bzw. BPEL-Dateien sich innerhalb eines Projektes befinden.

Die Palette ist erweiterbar. Es besteht die Möglichkeit in die Palette eigene Sektionen bzw. grafische BPEL-Elemente hinzuzufügen und somit die Menge von verfügbaren Elementen zu

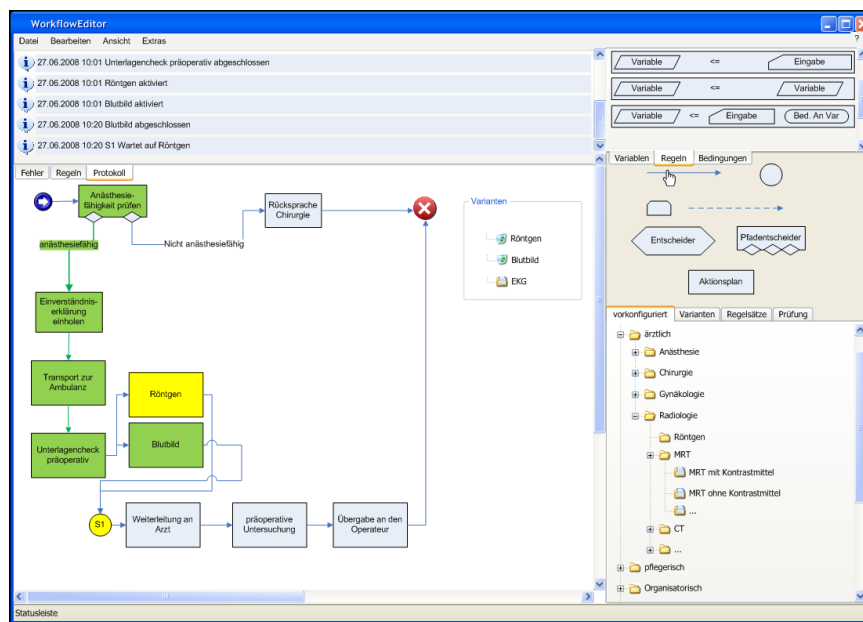


Abbildung 6.12: Designstudie: Die Synchronisation den Aktionsplänen durch Synchronisator.

erweitern. Als Beispiel kann man die Integration von neuen BPEL4People-Element nennen.

Der Designer unterstützt den Entwickler beim Erstellen der BPR-Datei, die man danach in die *ActiveBPEL*-Engine hochladen kann. Dafür wurde der ursprüngliche BPEL-Editor um eine Schaltfläche erweitert (s. Abbildung 6.20 auf Seite 136).

Nach dem BPR-Erstellungsvorgang (Abbildung 6.21) wird im Projekt ein Verzeichnis *bpr-build* erstellt, in dem sich die BPR-Dateien befinden (Abbildung 6.22).

Quelltextansicht

In der Quelltextansicht wird der Quelltext der geöffneten BPEL-Datei dargestellt. Die Sicht wurde im Wesentlichen nicht geändert. Der Quelltext von BPEL-Prozessen lässt sich manuell ändern und die Änderungen in der BPEL-Sicht sofort beobachten. Diese Sicht wurde zum Teil auch für die neuen Anforderungen, wie z. B. BPEL4People-Elemente, angepasst. Es können auch die neuen BPEL-Elemente im XML-Code betrachtet werden.

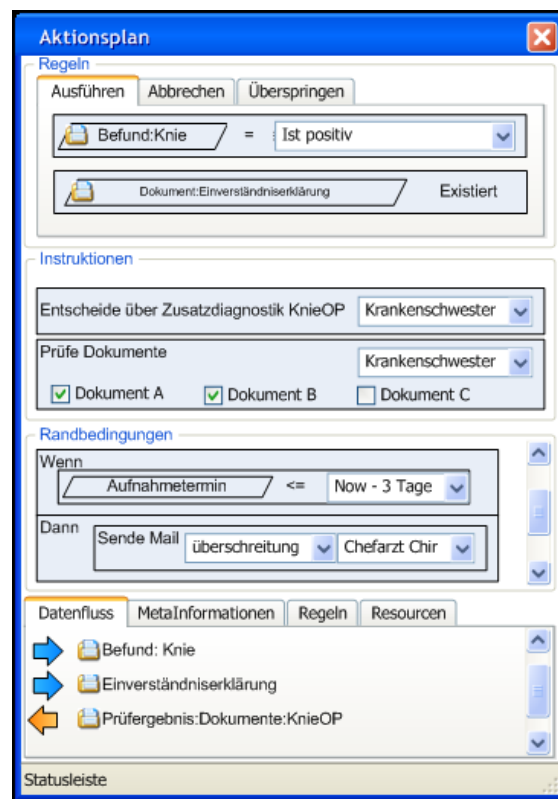


Abbildung 6.13: Designstudie: Die Darstellung eines Aktionsplanes im Editor.

6.2.1.2 Repository

Das Repository dient dazu, die XML-Dateien (z. B. WSDL- oder BPEL-Dateien) lokal in einer Datenbank zu speichern und zu verwalten. Dafür wurde von der PG das vorhandene Repository für BPEL verwendet und entsprechend angepasst. Die Dateien aus den BPEL-Projekten können per „drag-and-drop“ ins Repository kopiert werden, damit die Codebasis bzw. die Vorlagen verwalten werden können (Abbildung 6.24).

Die Dateien lassen sich im Repository in hierarchischen Strukturen abspeichern und danach mithilfe einer SQL-ähnlichen Sprache extrahieren.

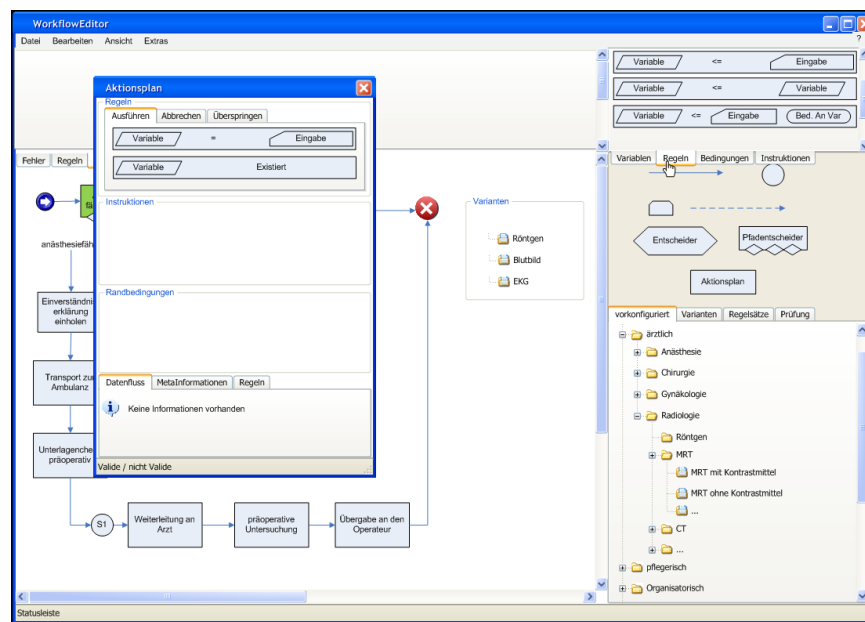


Abbildung 6.14: Designstudie: Das Editieren eines Aktionsplanes im Editor.

Suche

Das Repository stellt eine Suchfunktion zur Verfügung, d. h., dass die grafische Oberfläche („Graphical User Interface“ oder „GUI“) eine entsprechende Funktionalität haben und die Logik dahinter vorhanden sein muss.

Da die Information im Repository in einer hierarchischen Form abgelegt wird, ermöglicht es auch eine gute Strukturierung der Daten. Die Suche kann auf verschiedenen Ebenen gestartet werden, um die nicht relevanten Daten zu vermeiden.

Pflege

Das Repository bietet die Grundfunktionalität von Datenbanken, d. h. Objekte, die im XML-Format vorliegen, können leicht gespeichert und geladen werden.

Ad-hoc-Modul

Dient zur Kommunikation mit der Engine. Die Änderungen im Editor werden an die entsprechende Prozessinstanz geschickt und auf dem Server übernommen. Die Prozessinstanz kann nach mehreren Parametern durchsucht werden. Sobald die Instanz identifiziert ist, muss sie analysiert und der Prozessstatus ermittelt werden.

Die Änderungen aus dem Editor und die aktuelle Prozessinformation werden benötigt, um die Änderungen am Prozess vorzunehmen.

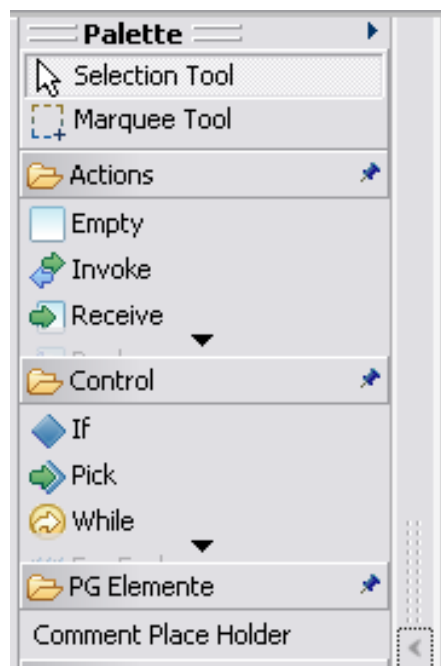


Abbildung 6.15: BPEL-Elemente in der Palette des Designers

6.3 Repository

Das Repository, welches wir für unsere Zwecke ausgewählt haben, ist das IBM BPEL Repository. Es wurde im Jahre 2006 von IBM alphaworks entwickelt um BPEL-Prozesse und andere XML-Dateien zu verwalten. Dabei bietet dieses Repository einfache Möglichkeiten um Daten, die von anderen Softwarelösungen erzeugt wurden, zu importieren und exportieren. Der Zugriff auf diese Daten erfolgt dabei in Form von Java-Objekten, was es natürlich für andere Java Programme einfacher macht, diese Daten zu verarbeiten. Es besteht auch die Möglichkeit Abfragen auf den gespeicherten Daten mithilfe einer Query Engine zu machen. Einmal bietet IBM hierfür eine Lösung an, die Object Constraint Language (OCL), aber es gibt auch eine Schnittstelle, so dass man auch andere Query Engines einbinden kann.

Einer der entscheidenden Vorteile dieses Repository ist die Tatsache, dass es relativ einfach mit neuen Datenmodellen erweitert werden kann. Diese können aus XML-Schemen, UML-Klassendiagrammen oder aus Java-Code erzeugt und im Repository eingebunden werden.

6.3.1 Überblick Architektur

In Abbildung 6.25 bekommt man einen Überblick über die Zusammenhänge und Schnittstellen des Repositories. Die Hauptkomponente des Repositories ist dabei das Repository

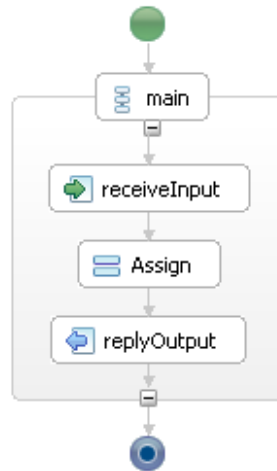


Abbildung 6.16: Darstellung eines BPEL-Prozesses im Designer

```

<bpel:partnerLinks>
  <bpel:partnerLink name="echoPLT" partnerLinkType
</bpel:partnerLinks>

<!-- =====
<!-- ORCHESTRATION LOGIC
<!-- Set of activities coordinating the flow of mess
<!-- services integrated within this business proces
<!-- =====

<bpel:variables>
  <bpel:variable name="echoRequest" messageType="t
  <bpel:variable name="echoResponse" messageType="
</bpel:variables>
<bpel:sequence name="main">

```

Abbildung 6.17: Quelltext eines BPEL-Prozesses im Editor

ry Application Programming Interface (API), welches die Daten verwaltet. Diese Daten werden in sogenannten Organisationen in Form einer Baumstruktur gespeichert. Jede dieser Organisationen kann Dateneinheiten enthalten, die als XML-Dateien importiert und exportiert werden können. Die API verwaltet auch die Abfragen, die auf den gesamten Organisations-Baum oder einen Teilbaum angewendet werden. Gleichzeitig bietet die API noch Schnittstellen bzw. Erweiterungspunkte um das Repository mit neuen Query Engines oder Datenmodellen, die mit dem Eclipse Modelling Framework (EMF) erstellt werden können, zu erweitern.

Die in Abbildung 6.25 unterschiedlich gewählten Farben sollen die verschiedenen Kategorien voneinander trennen. Die blauen Komponenten werden z. B. von Eclipse bereitgestellt. Die gelben Elemente stehen für die Teile, die für das Repository konstruiert wurden. Der rote Block (Kent OCL Engine) beschreibt eine bestehende externe Komponente, die mit dem Repository zusammenarbeitet. Letztlich die grünen Elemente, die die verschiedenen Möglichkeiten zeigen, wie externe Programme das Repository erweitern bzw. nutzen können.

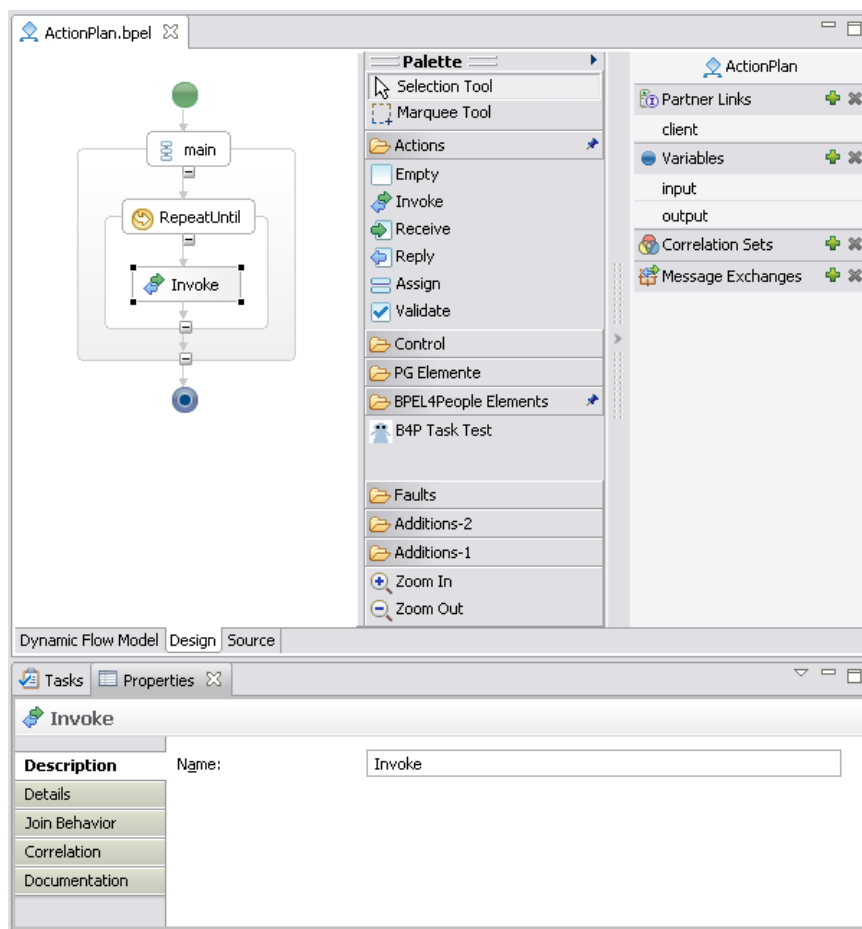


Abbildung 6.18: Die BPEL-Sicht des Designers.

Abbildung 6.26 zeigt den Aufbau einer einzelnen Organisation. Es existiert eine Descriptor XML-Datei in jeder Organisation, die die Beziehungen der unterschiedlichen XML-Dateien zu einer BPEL-Datei im Organisations-Verzeichnis herstellt. Für jede Datei in der Organisation werden in der Descriptor-Datei folgende Daten spezifiziert:

- **File Name** identifiziert die Datei in ihrer Organisation
- **File Type** spezifiziert den Typ der Datei. Die Anzahl bestimmter Dateitypen ist pro Organisation beschränkt (vergleiche Kantenbeschriftung in Abbildung 6.26).
- **File Content Type** wird benötigt, um Dateien, die in der gleichen Organisation und vom gleichen Typen sind, aber unterschiedliche Rollen bzw. oft auch unterschiedliche EMF-Modelle haben, zu identifizieren. Hierbei wird ein EMF-Modell auf ein bestimmtes XML-Schema abgebildet.

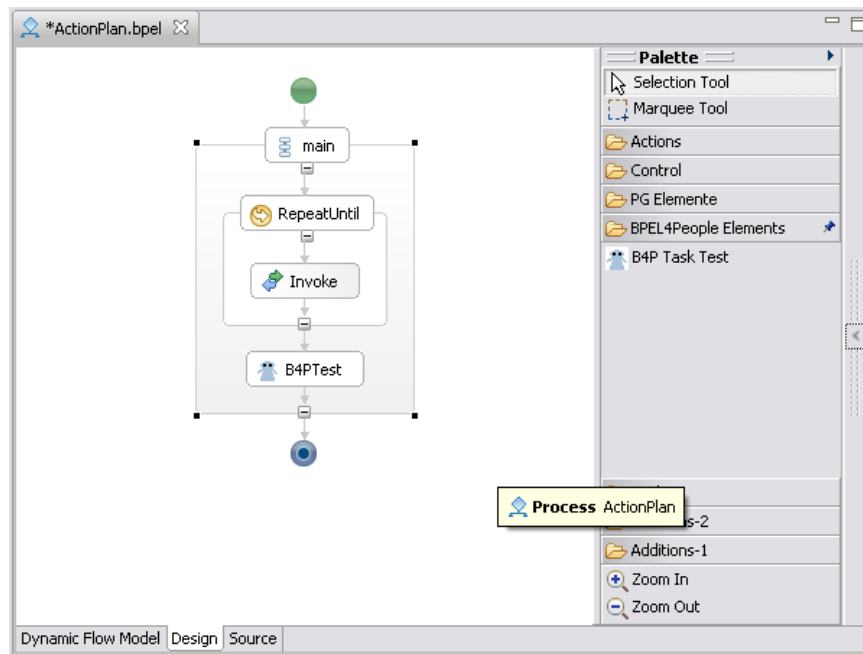


Abbildung 6.19: Das Beispiel eines neuen BPEL-Konstruktes

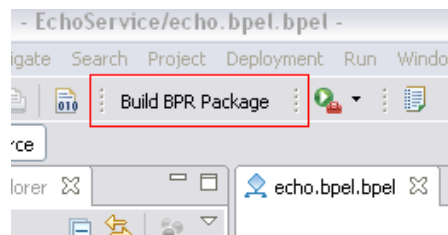


Abbildung 6.20: Die Schaltfläche zur Erstellung der BPR-Datei aus einem Projekt

- **URI** identifiziert den Ort, an dem die entsprechende Datei abgelegt ist.

6.3.2 Repository als Container

Das BPEL Repository bietet den Zugang zu den Objekten an, die aus XML-Dateien geladen wurden, d.h. es bietet die Möglichkeit an, dass Programme die Repository API benutzen können, um direkt auf die Java Objekte zuzugreifen. Die Datenpersistenz des Repository wird mit Hilfe des Eclipse Modeling Framework (EMF) abgewickelt. Das ist ein Eclipse-Plugin, welches speziell zum Laden und zum Speichern von Objekten in XML Metadata Interchange (XMI) und XML-Formaten entwickelt wurde. Die Modellsprache, die benutzt wird, um Modelle in EMF darzustellen, ist Ecore. Das Ecore Modell kann aus UML, Java oder XML-Schemata erstellt werden und EMF Java Code kann wiederum aus einem

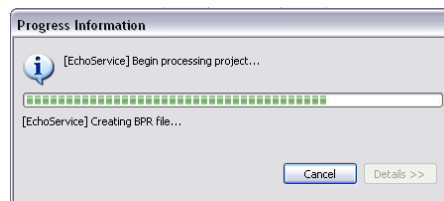
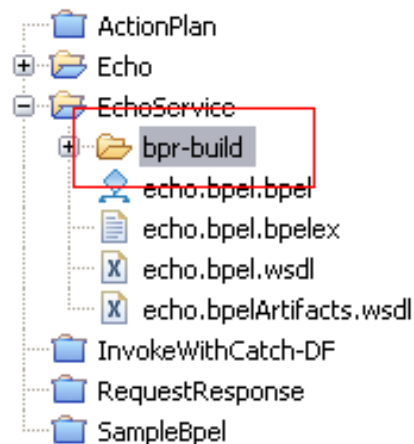


Abbildung 6.21: Die Erstellung der BPR-Datei aus einem Projekt

Abbildung 6.22: Das *bpr-build*-Verzeichnis in einem Projekt

EMF Ecore Modell erstellt werden. Wenn der EMF Java-Code erzeugt wird, wird die EMF-Funktion für die Datenpersistenz dazu gegeben. Wie die Abbildung 6.27 [101] zeigt, gibt es noch die Möglichkeit, dass die Klassen von Java EMF Modellen automatisch aus dem Ecore Modell erstellt werden. Mit der eingebetteten Funktion unterstützen die Klassen den EMF-Datenpersistenz-Mechanismus. Das Ecore Modell kann aus Java Code für die Klassen, IBM Rational Rose UML-Modellen und XML-Schemata automatisch erzeugt werden. EMF hilft die Objekt-Modelle für BPEL, WSDL und XML-Schema aufzubauen. BPEL Repository unterstützt den Query-Mechanismus, mit dem das Dateien-Objekt-Modell im Repository abgefragt werden kann. Die abgefragte Datei wird als ein EMF-Objekt mit Hilfe der Query-Engine durch die Query-Parameter geladen und als Ergebnis zur API zurückgegeben (siehe Abbildung 6.28) [101]. Es gibt sechs Parameter mit den Buchstaben A-F, (vgl. Abbildung 6.28). Der Parameter Query Scope definiert die Organisationen, in denen die Anfrage ausgeführt wird. Er beschränkt den Bereich für die Anfrage aus dem Repository, sodass nur in diesem Bereich die Anfrage ausgeführt wird. Der boolesche Parameter Include Suborganizations definiert, ob der in Query Scope definierte Anfragenbereich die jeweils enthalten Unterordner mit einschließt. Der Parameter für Query File Type und Query Content Type beschränkt die in der Organisation enthaltenden Dateien, die angefragt werden. Die oberen vorgestellten drei Parameter entscheiden, welche Dateien als EMF Objekte zur

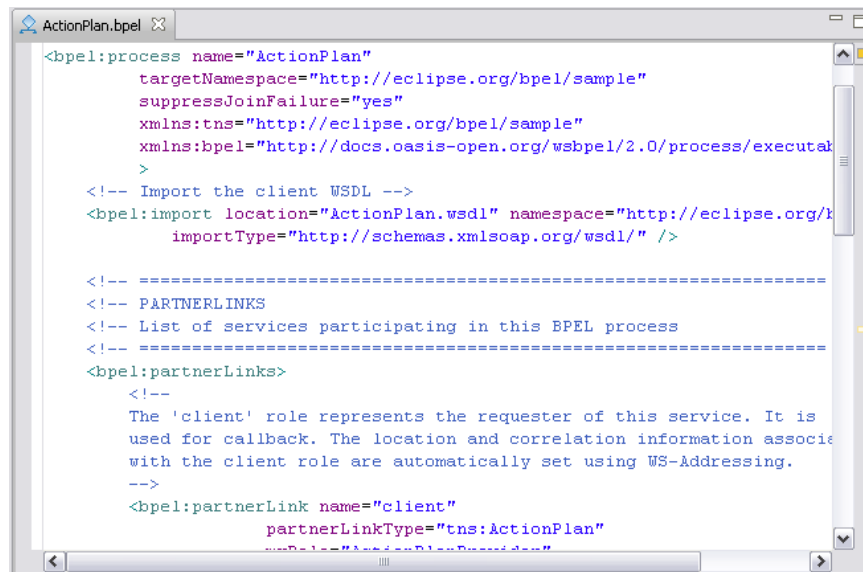


Abbildung 6.23: Die Quelltextansicht des Designers.

Query-Engine geschickt werden. Der Object Constraint Language (OCL) Query String hilft, die Engine für die Anfrage mit jedem EMF Objekt auszuführen. Dieser Parameter enthält eine Reihe von Parametern für die Query-Engine. Der Parameter Return File Type und Return Content Type definiert, welche Datei mit einer positiven Antwort zurückgegeben wird. Der Parameter Return EMF Modell entscheidet, was in die Antworttabelle gestellt wird, wenn die Antwort positiv ist (eine boolesche Variable oder ein EMF Objekt). Die Parameter Return File Type inklusive Return Content Type und Return EMF Modell entscheiden sich für entsprechenden Ausgaben. [101]

6.3.3 Erweiterung von Modellen

Wie bereits weiter oben erwähnt, ist einer der großen Vorteile dieses Systems, dass es sich relativ einfach mit neuen EMF-Modellen erweitern lässt. Diese Erweiterungen können z. B. in Form von UML-Klassendiagrammen importiert werden. Die EMF-Modelle werden standardmäßig als *metadata* abgelegt, könnten aber auch *.xml* heißen, da sie mit jedem Typ von XML-Datei, die ein EMF-Modell hat, benutzt werden können.

In unserem Fall sollen sowohl Elemente aus der technischen Sicht (BPEL) als auch aus der fachlichen Sicht (Aktionspläne usw.) im Repository abgelegt bzw. verwaltet werden. Da es z. B. unterschiedliche Arten von Aktionsplänen oder Instruktionen gibt, d.h. mit unterschiedlichen Voraussetzungen, Variablentypen usw., können hier die entsprechenden Modelle erzeugt werden. Im Folgenden wird anhand eines einfachen Beispiels kurz erläutert, wie diese Erweiterung aus Java-Code eingefügt wird.

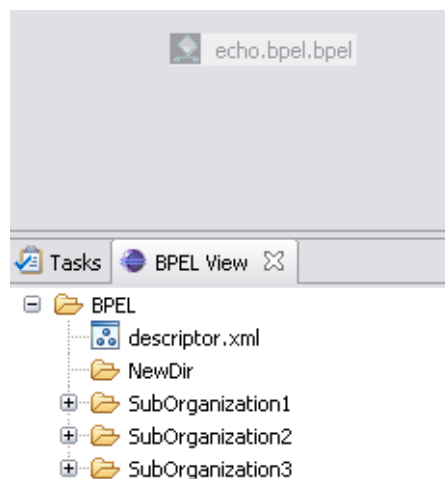


Abbildung 6.24: Das Kopieren einer Datei ins Repository

- Zunächst muss in Eclipse *new workspace* und als Perspektive *Java* ausgewählt werden.
- Im nächsten Schritt gibt es, wie bereits erwähnt, drei Varianten um ein EMF Ecore Modell zu erzeugen. Für dieses Beispiel muss die *Java annotations*-Variante gewählt werden.
 - Nun muss Java Projekt in den aktuellen *workspace* importiert werden.
 - Anschließend mit einem Rechtsklick auf das *Java package* (z.B. *emfsample*) und ein neues EMF Modell auswählen. Hierbei muss der Ordner gewählt werden in dem der Java-Code des zu importierenden Teils liegt. Nun wird der Dateiname in *emfsample.genmodel* geändert.
 - Jetzt ist *Load from annotated Java* auszuwählen und *next* zu drücken.
 - Im folgenden Schritt wird das *package*, das man in das Modell eingebunden werden soll, auszuwählen und gegebenenfalls den *Core Model Name* in *emfsample.ecore* umzubenennen.
 - Jetzt können noch die *ecore* Dateieigenschaften geändert und die Namen verkürzt werden, um z. B. die XML-Dateien übersichtlicher zu gestalten. Dazu muss die *ecore*-Datei geöffnet und mit einem Rechtsklick „Eigenschaften“ ausgewählt werden.
 - Anschließend müssen die Eigenschaften der *genmodel*-Datei geändert werden, indem hier die entsprechenden Pfade für *Model Plug-in*, *Edit Directory*, *Editor Directory*, *Edit Plug-in Class*, *Editor Plug-in Class* und *Model Directory* gesetzt werden. Abschließend ist alles zu speichern.

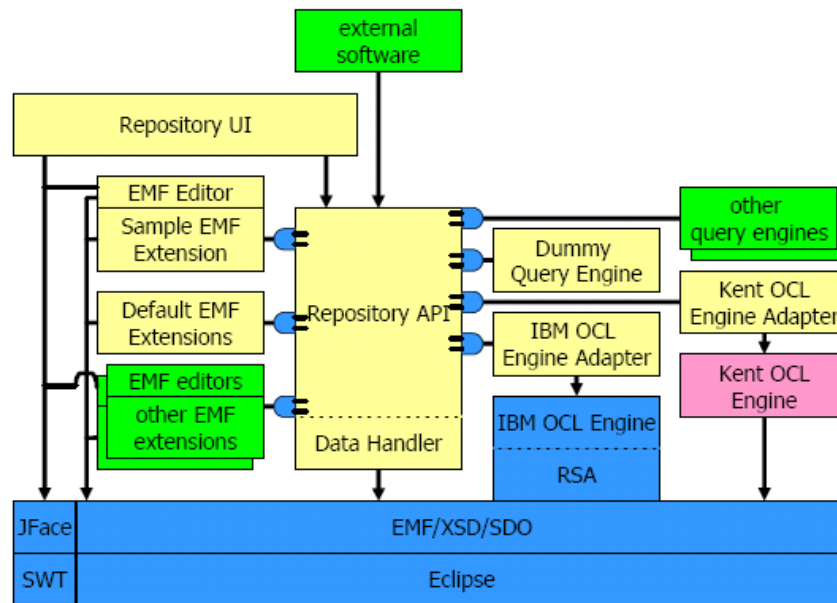


Abbildung 6.25: Repository Aufbau .

- Aus dem zuvor erzeugten EMF Ecore Modell soll automatisch Java-Code generiert werden.
 - Dazu wird ein neues Java Projekt geöffnet und z. B. EMF-Sample genannt. In dem gleichen Dialog muss noch *Create separate source and output folders* ausgewählt werden.
 - Anschließend muss in dem EMF-Sample Projekt ein neuer Ordner (z. B. *emf*) erzeugt werden.
 - In diesen Ordner werden die zuvor erzeugten *ecore* und *genmodel* Dateien eingefügt.
 - Jetzt muss noch die eingefügte Datei (in diesem Beispiel sollte es *emfsample.genmodel* sein) geöffnet werden und mit einem Rechtsklick in der Editoransicht *Generate Model Code* ausgewählt werden. Anschließend wird automatisch der entsprechende Java-Code erzeugt.
- Mit einigen weiteren Schritten wird aus diesem erzeugten Java-Code eine EMF Erweiterung durch erweiteren der *AbstractEMFExtension* Klasse erzeugt.
- Schließlich kann daraus noch ein EMF-Editor kreiert werden, der Dateien aus EMF Modellen erzeugt.
- Abschließend muss diese erzeugte Datei in das Repository übernommen werden. Dazu muss wieder in die BPEL Repository Perspektive gewechselt werden. Hier ist das

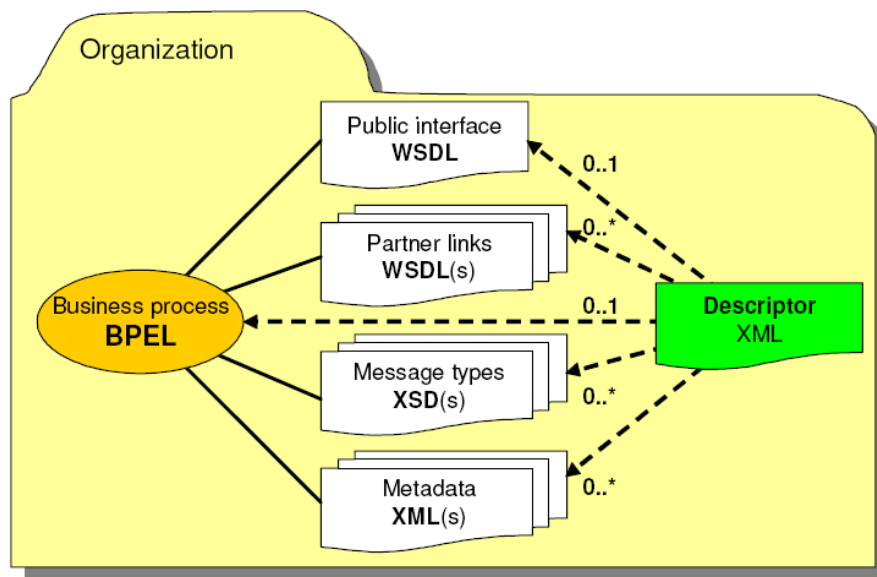


Abbildung 6.26: Schema einer Repository Organisation .

Repository zu öffnen und die Datei zu importieren, indem über einen Rechtsklick auf das Repository, die Datei ausgewählt und als *New File Content Type* den zuvor als Erweiterung der *AbstractEMFExtension* Klasse erzeugten Typ gesetzt wird.

Das Ergebnis dieser Schritte ist in Abbildung 6.29 gezeigt, wobei die erzeugte Datei als *metadata* in die Organisation übernommen wurde.

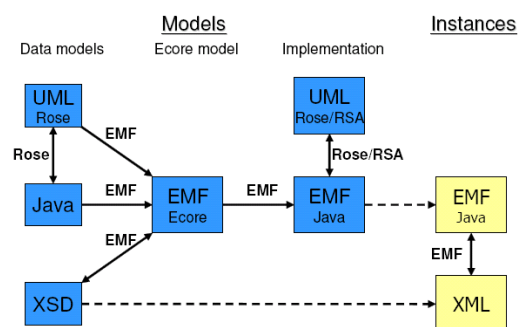


Abbildung 6.27: Modell- und Datenformattransformation mit Eclipse Modeling Framework

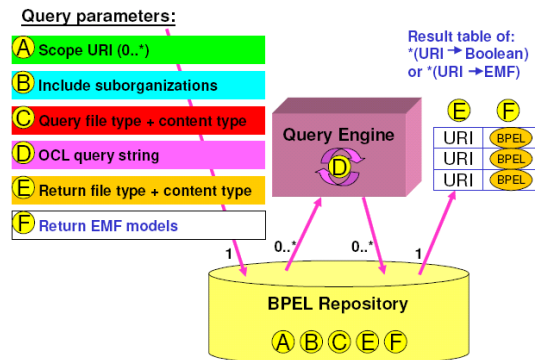


Abbildung 6.28: Query Mechanismus des Repositories .

6.3.4 Umsetzung

Zur Einbindung des BPEL Repositories in die speziell entwickelten Editoren, wurde eine eigene View hinzugefügt. Die Elemente werden in einer Verzeichnisstruktur dargestellt, wobei es hier zwei Kernmerkmale gibt. Zum einen können die Elemente der fachlichen Sicht (Aktionspläne, Instruktionen etc.) dargestellt werden, wenn man sich im entsprechenden Editor befindet (siehe Abbildung 6.30), und zum anderen werden die BPEL Elemente für den technischen Editor separat verwaltet und auch angezeigt (siehe Abbildung 6.31).

Neu angelegte BPEL-Prozesse können per „drag-and-drop“ in das Repository gezogen und so gespeichert werden. Außerdem gibt es Funktionen zum Hinzufügen und zum Entfernen von Ordnern, sodass die Struktur beliebig erweitert werden kann.

6.4 Interaktion zwischen Engine und Designer

Zum Deployment neuer Workflowdefinitionen, zum Monitoring laufender Prozessinstanzen und zur Änderung existierender Instanzen war es notwendig, eine Kommunikationsmöglichkeit zwischen der Designerkomponente und der ausführenden Engine bereit zu stellen. Im Folgenden gehen wir nun auf die von der Engine zur Verfügung gestellten Operationen zum Deployment, Monitoring und Modifikationen ein.

Während aus Sicht des Modellierers/Bearbeiters eine Unterscheidung zwischen der Anwendung von Varianten auf laufenden Prozessinstanzen und der Durchführung von ungeplanten Ad-hoc-Modifikationen besteht, können aus Sicht der Engine beide Änderungstypen durch Ad-hoc-Modifikationen der Workflowinstanz durchgeführt werden. Aus diesem Grund sind im Folgenden unter Modifikation immer sowohl die Anwendung von Varianten, als auch echte Ad-hoc-Modifikationen zu verstehen.

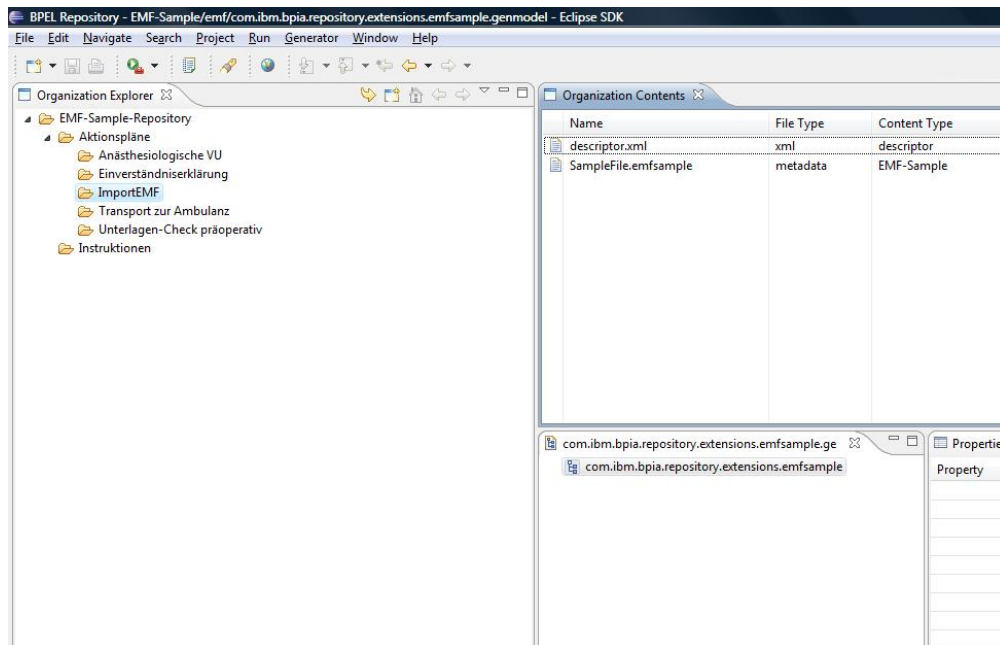


Abbildung 6.29: Beispiel eines neu importierten EMF Modells .

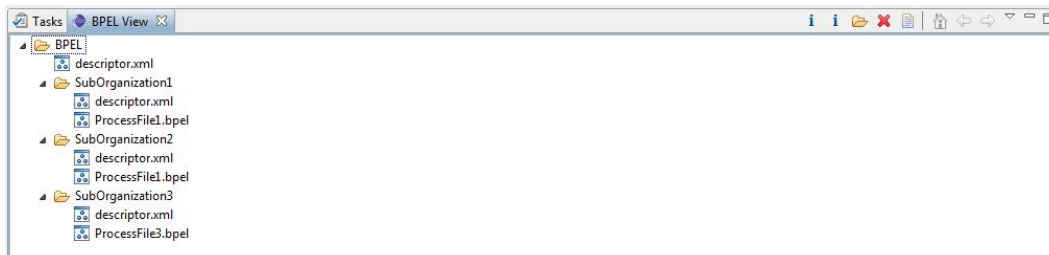


Abbildung 6.30: Repository View - Technische Ansicht .

6.4.1 Deployment

Unter Deployment versteht man das Einbringen und Verfügbarmachen einer neuen Prozessdefinition in BPEL-Engines. Eine Prozessdefinition besteht im Wesentlichen aus einer Prozessbeschreibung in Form von BPEL-Code, Schnittstellenbeschreibungen in WSDL-Dateien und XML-Schema Dateien in Form von XSD-Dateien. Zusätzlich benötigt die BPEL-Engine allerdings weitere Metainformationen (siehe hierzu [6.1.3](#)). Da für das Deployment von BPEL-Prozessen (noch) kein einheitlicher Standard existiert, geht jeder Hersteller von BPEL-Engines einen eigenen Weg. Bei ActiveBPEL werden alle Prozessartefakte und Metainformationen in einem Archiv mit der Dateiendung „.bpr“ zusammengefasst (mehr zum Aufbau der Archivdatei findet sich in [6.1.3](#)). Für diese Archivdatei kann auf zwei verschiedene Arten in das Deployment in der BPEL-Engine durchgeführt werden:

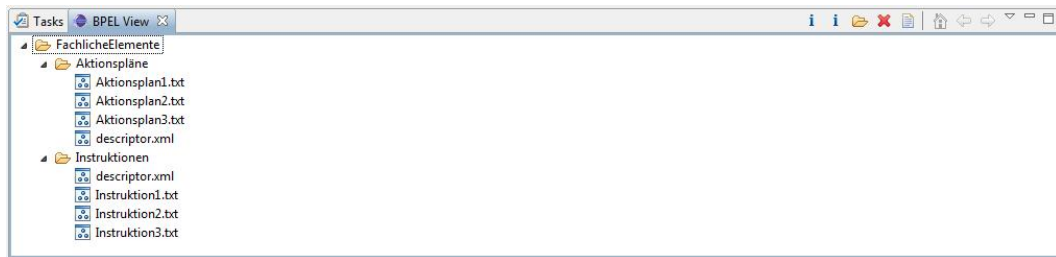


Abbildung 6.31: Repository View - Fachliche Ansicht .

- Dateisystem-Deployment
Die einfachste Form des Deployments besteht im simplen Kopieren des BPR-Archivs in einen festgelegten Ordner der laufenden Engine-Instanz. Durch einen Hot-Deployment-Mechanismus erkennt die Engine neue und geänderte Prozessdefinitionen und führt das Deployment durch.
- Web Service Deployment
Eine weitere Möglichkeit zum Deployment stellt die Nutzung eines speziellen Web Service dar, der von der Engine angeboten wird. Hierzu wird die Operation *deployBPR* des Web Service mit dem Base64-codierten BPR-Archiv und dessen Namen aufgerufen. Die Engine führt daraufhin das Deployment durch, und antwortet mit einer Statusmeldung oder, wenn das Deployment nicht erfolgreich durchgeführt werden konnte, mit einem SOAP Fault. Anschließend platziert die Engine das Archiv im Dateisystem, so dass bei einem Neustart der Engine das oben beschriebene dateisystembasierte Deployment durchgeführt werden kann.

Da davon ausgegangen werden kann, dass Engine und Designer im Normalfall nicht auf dem selben Computer laufen, und somit kein direkter Zugriff auf das Dateisystem der Engine möglich ist, wird von der PG zum Deployment neuer Workflows durch den Designer ausschließlich der Web Service genutzt werden. Ein weiterer Vorteil dieses Vorgehens ist, dass auftretende Fehler während des Deployments direkt an den Designer (in Form eines SOAP Faults) zurückgegeben werden können. Bei der Nutzung des dateisystembasierten Deployments wäre hingegen eine aufwändige Überwachung und Auswertung von Logdateien notwendig, um festzustellen, ob das Deployment erfolgreich durchgeführt werden konnte.

6.4.2 Monitoring

Unter Monitoring verstehen wir im Folgenden das Abfragen des Zustands von Workflowinstanzen. Monitoring ist einerseits sinnvoll für Bearbeiter, um sich über das Fortschreiten der Behandlung des Patienten zu informieren. Andererseits ist das Monitoring eine notwendige Voraussetzung, um eine Änderung von laufenden Prozessinstanzen durchführen zu können. Das Monitoring gliedert sich in zwei Phasen

- Finden von Instanzen
- Anzeigen des Status von Instanzen

Wir gehen zunächst darauf ein, wie der Designer laufende, bzw. bereits abgeschlossene Prozessinstanzen finden kann und zeigen anschließend, wie der Zustand von konkreten Prozessinstanzen im Designer angezeigt werden kann.

Finden von Prozessinstanzen

Der standardmäßig in ActiveBPEL verfügbare Administrationswebservice ermöglicht bereits eine Suche nach Prozessinstanzen über die Operation *getProcessList*. Dabei kann nach zahlreichen Kriterien wie ProzessID, Prozessname, Start- und Enddatum gesucht werden. Ein (verkürztes) Beispiel für das Ergebnis einer solchen Anfrage zeigt Listing 6.9.

Listing 6.9: getProcessList Ergebnis (gekürzt)

```
<rowDetails>
  <item>
    <ended>2008-07-08T08:51:00.000Z</ended>
    <name xmlns:ns1="http://example">ns1:example</name>
    <processId>4201</processId>
    <started>2008-07-08T08:50:42.000Z</started>
    <state>3</state>
    <stateReason>-1</stateReason>
  </item>
  <item>
    <ended>2008-07-01T13:13:38.000Z</ended>
    <name xmlns:ns2="http://SimpleBPEL">ns2:SimpleBPEL</name>
    <processId>4101</processId>
    <started>2008-07-01T13:13:36.000Z</started>
    <state>1</state>
    <stateReason>-1</stateReason>
  </item>
</rowDetails>
```

Für den Fachanwender ist eine Suche nach Prozessinstanzen auf diesem Wege allerdings nicht praktikabel. Er ist vielmehr daran interessiert sich alle Prozesse für einen bestimmten Patienten anzeigen zu lassen. Aus diesem Grund unterstützt der von der PG implementierte Web Service, zusätzlich zu den oben genannten Filtermöglichkeiten, die Filterung nach einer eindeutigen PatientenID. Technisch wird dies dadurch realisiert, dass jede neue Prozessinstanz die PatientenID als BPEL-Variable speichert. Eine andere angedachte Lösung bestünde darin, ein Mapping zwischen PatientenIDs und zugehörigen Prozessinstanzen von der Engine verwalten zu lassen. Dies hätte allerdings einen grösseren Eingriff in die interne Architektur der Engine erfordert. Da auf Änderungen der Engine möglichst verzichtet werden sollte, wurde diese Lösung verworfen, auch wenn sie vermutlich effizienter gewesen wäre.

Anzeige des Zustands von Prozessinstanzen

Nachdem der Benutzer mittels *getProcessList* eine Liste aller vorhandenen Prozessinstanzen erhalten hat, hat er die Möglichkeit einzelne Prozessinstanzen im Detail zu betrachten. Auch dafür existiert im ActiveBPEL Web Service bereits eine Implementierung: *getProcessState()* liefert Details für eine bestimmte ProzessID. Listing 6.10 zeigt dafür ein (verkürztes) Beispiel.

Die Rückgabe dieser Operation wurde von der PG allerdings grundlegend geändert: Problematisch in der originalen Implementierung ist zum Beispiel, dass der Zustand von BPEL Aktivitäten in Form eines Integers, der einem internen Zustand der ActiveBPEL Engine entspricht angezeigt wird. Der Fachanwender kann mit diesen Zuständen allerdings nichts anfangen. Es ist auch nicht direkt ersichtlich, welcher Integerwert welchen Zuständen entspricht, da diese Information nur in dem Quelltext der Engine vorhanden ist. Daher wird hier ein Mapping der internen Zustände der Engine auf die von Team A definierten Zustände (NOT_ACTIVATED, ACTIVATED,...) durchgeführt. Ein weiteres Problem besteht in der Form der Darstellung der Zustände. Um eine graphische Repräsentation innerhalb des Editors vornehmen zu können, müsste der zurückgegebene Ausdruck aufwändig geparkt werden. Da, wie in Kapitel 4.3 beschrieben, alle BPEL-Aktivitäten mit einer prozesseindeutigen ID versehen sind, kann die Darstellung des Zustandes einfach erreicht werden, indem ein Mapping von ID zu Zustand zurückgegeben wird.

Ein bisher nicht angesprochenes Problem ist, auf welche Weise der Editor eigentlich wissen soll, wie die Prozessdefinition konkret aussieht. Es muss davon ausgegangen werden, dass der angezeigte Prozess bereits durch Einfügen einer Variante, oder durch eine Ad-hoc-Modifikation verändert wurde. Da im Repository solche Prozesse nicht abgelegt werden sollen, muss eine andere Lösung gefunden werden. Die PG hat sich dafür entschieden zusätzlich zu benötigten BPEL-Definitionen alle weiteren Modellbeschreibungen innerhalb des BPR-Archivs abzulegen und für jede Prozessinstanz zu verwalten. Durch einen Aufruf von *getProcessState()* werden dann nicht nur Zustandsbeschreibungen sondern auch das komplette BPR-Archiv in Base64 kodiert zurückgegeben. Mit den Informationen aus dem BPR-Archiv kann der Editor eine genaue Prozessrepräsentation rekonstruieren. Die Details dieser zusätzlichen Implementierungen sollen hier nicht weiter erläutert werden, da dies hauptsächlich Codepassagen sind, die aus der Nutzung von AXIS und Base64 Kodierung resultierten.

Listing 6.10: *getProcessState* Ergebnis (gekürzt)

```
<bpelObject currentState="Finished"
  locationPath="/process/sequence" terminating="false">
  <bpelObject currentState="Finished"
    locationPath="/process/sequence/receive[@name='Rec']"
    queued="false" terminating="false"/>
  <bpelObject alarmId="-1" currentState="Finished"
    locationPath="/process/sequence/WAIT" queued="false"
    terminating="false"/>
  <bpelObject currentState="Finished"
```

```

    locationPath="/process/sequence/flow" terminating="false">
  <bpelObject currentState="Finished"
    locationPath="/process/sequence/flow/assign[@name='Assign2']"
    terminating="false"/>
  <bpelObject currentState="Finished"
    locationPath="/process/sequence/flow/assign[@name='Assign1']"
    terminating="false">
    <link currentState="Inactive" evaluated="true"
      locationPath="/process/sequence/flow/links/link[@name='L1']"
      terminating="false"/>
  </bpelObject>
</bpelObject>
<bpelObject currentState="Finished"
  locationPath="/process/sequence/reply[@name='Rep']"
  terminating="false"/>
</bpelObject>

```

6.4.3 Modifikationen

Modifikationen durch den Benutzer zur Laufzeit einer Prozessinstanz sind nur auf Ebene der Workflows möglich. Es lassen sich also z. B. Aktionspläne löschen, verschieben, und hinzufügen. Es ist aber nicht möglich in die interne Struktur eines Aktionsplans während der Laufzeit der Prozessinstanz einzugreifen. Aktionspläne sind vom Benutzer also als atomar zu betrachten. Da der Workflow auf BPEL-Ebene im Wesentlichen aus einem großen *Flow* besteht, in dem der Aufruf von Aktionsplänen durch *invokes* und der Kontrollfluss zwischen Aktionsplänen durch *links* mit entsprechend gesetzten *transitionCodes* und *joinConditions* realisiert wird, lassen sich Änderungen durch das Hinzufügen und Löschen von *invokes* und *links* umsetzen.

Jede Änderung, die vom Bearbeiter durchgeführt wird, führt zu einer Modifikation des BPEL-Codes des Workflows. Mit Hilfe des BPR-Builders (Kapitel 6.1.3) wird anschließend durch den Editor ein neues BPR-Archiv erzeugt und an den Engine Web Service geschickt. Die Engine führt dann die notwendigen Operationen durch, um die gewünschte Modifikation zu realisieren. Eine detaillierte Beschreibung der Operationen für das Ad-hoc-Umfeld wird in 6.5 gegeben.

Damit eine Ad-hoc-Modifikation erfolgreich durchgeführt werden kann, muss die zu modifizierende Instanz zunächst angehalten, und erst nach erfolgreicher Änderung wieder aufgenommen werden. Dies wird im folgenden Kapitel beschrieben.

6.4.4 Suspend/Resume

Abbildung 6.32 zeigt eine beispielhafte Visualisierung des Ablaufs einer Prozessinstanz. Wir nehmen nun an, dass während die Aktivität „Unterlagencheck präoperativ“ durchgeführt wird, eine Ad-hoc-Modifikation durchgeführt werden soll. Der Bearbeiter möchte die

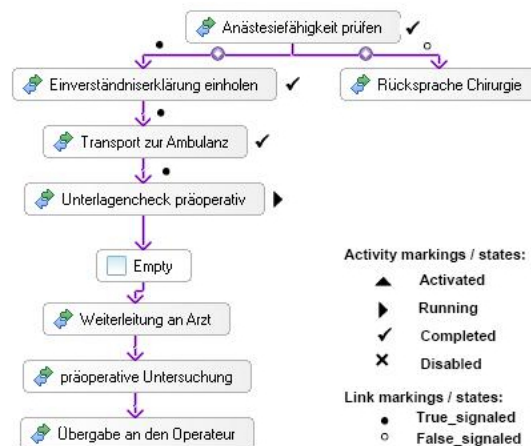


Abbildung 6.32: Zustand einer Prozessinstanz .

Aktivitäten „Röntgen“ und „Blutbild“ so einfügen, dass sie direkt nach Abschluss von „Unterlagencheck präoperativ“ und vor „Weiterleitung an Arzt“ durchgeführt werden. Würde man die Prozessinstanz vor dieser Änderungsoperation nicht anhalten, könnte es passieren, dass „Unterlagencheck präoperativ“ abgeschlossen wird, und damit „Weiterleitung an Arzt“ ausführbar wird. Würde der Bearbeiter nun versuchen, die durchgeführten Änderungen an die Engine zu kommunizieren, müsste diese dies ablehnen, da die Änderung scheinbar in der Vergangenheit geschieht. Aus diesem Grund müssen alle Prozessinstanzen, die geändert werden sollen, durch Aufruf von *suspendProcess()*, vorher angehalten werden. Dadurch wird auch vermerkt, dass die Instanz gerade modifiziert wird, sodass es nicht zu konkurrierenden Änderungswünschen unterschiedlicher Bearbeiter kommen kann.

Die Operation *suspendProcess()* ist ebenfalls bereits im ActiveBPEL Web Service vorhanden. Allerdings ist die vorhandene Implementierung für unsere Anforderungen, wie im Folgenden gezeigt wird, nicht ausreichend. Wir geben nun eine kurze Einführung in die Funktionsweise von *suspendProcess()*.

suspendProcess()

Ein Aufruf von *suspendProcess* führt dazu, dass die Executionqueue von ActiveBPEL in den suspended Zustand versetzt wird. Dies bedeutet, dass keine neuen BPEL-Aktivitäten gestartet werden können. Bereits laufende Aktivitäten können und werden allerdings noch abgeschlossen. Dies ist natürlich auch sinnvoll: Man denke z. B. an den Aufruf eines externen Web Service mittels *invoke*. Das *invoke* hat den externen Web Service aufgerufen, und wartet nun auf eine Antwort. Bevor die Antwort ankommt, wird *suspendProcess()* aufgerufen. Es wäre nun nicht sinnvoll diese Antwort nicht anzunehmen, weil die Instanz angehalten ist. Stattdessen sollte die Antwort angenommen und die *invoke* Aktivität beendet werden. Allerdings darf nun keine nachfolgende Aktivität gestartet werden, bis der Benutzer durch einen Aufruf von *resumeProcess()* die Ausführung fortsetzt.

Die existierende Implementierung von `suspendProcess()` realisiert genau dieses Verhalten. Problematisch für unsere Anwendung ist aber der Zeitpunkt, an dem die Executionqueue gefüllt wird: Wenn eine BPEL-Aktivität beendet wird, während sich die Instanz im Zustand „suspended“ befindet, wird direkt im Anschluss der Status aller ausgehenden Links aktualisiert. Daraufhin werden die nun ausführbaren Aktivitäten ermittelt und in die Executionqueue eingehängt. Wird nun eine Ad-hoc-Modifikation durchgeführt, müsste eventuell die Queue wieder geleert werden, weil einzelne Aktivitäten nun überhaupt nicht mehr ausführbar wären. Da die Umsetzung dieser Lösung nicht trivial und auch nicht effizient ist, wird von der PG ein anderer Ansatz gewählt: Aktivitäten können zwar im suspended Zustand beendet werden, die Auswertung von ausgehenden Links und die Bestimmung von Nachfolgeaktivitäten erfolgt allerdings erst, wenn der Prozess wieder fortgesetzt wird. Dazu musste die vorhandene Implementierung von `suspendProcess()` und `resumeProcess()` angepasst und erweitert werden.

Wie auch im Abschnitt 6.4.2 soll hier nicht weiter auf die technischen Details der Webserviceumsetzung bezüglich AXIS eingegangen werden. Jedoch wurden in diesem Zusammenhang wesentliche Anpassungen an der Engine durchgeführt. Diese für die „suspendProcess“ Operation relevanten Passagen werden im folgenden erläutert.

Die Nutzung eines Prozesses als Ressource wird in ActiveBPEL über eine Mutex-Implementierung gesteuert. Die Klasse „AeProcessWrapper“ umschließt einen Prozess und hält dann auch den Mutex für diesen Prozess. Wenn wir also den Wrapper eines Prozesses halten, halten wir auch den Mutex des Prozesses.

In der Klasse „AePersistentProcessManager“ wurde dieser vorhandene Mechanismus genutzt und bietet nun die Methoden „dfReleaseProcessWrapperWithMutex“ und „writeProcessAndGetMutex“ an, die diese Funktionalität öffentlich verfügbar machen.

Innerhalb von „writeProcessAndGetMutex“ wird die Methode „cancelReleaseTimer“ aufgerufen, die bewirkt, dass der Prozess persistent in die Datenbank geschrieben wird und danach nicht mehr im Speicher der Engine verfügbar ist. Anschließend wird für den übergebenen Prozess der Mutex durch eine Instanz der Klasse „AeProcessWrapper“ geholt und zurückgegeben. Weil keine anderen Operationen nun den Mutex des Prozess bekommen können, ist der Prozess angehalten. Die Methode „dfReleaseProcessWrapperWithMutex“ gibt den Prozess entsprechend wieder frei.

Nur der Thread, der den Mutex hält, kann diesen auch wieder freigeben. Da jedoch jeder Webservice-Aufruf über AXIS seinen eigenen HTTP-Thread erzeugt, wurde von der PG ein einzelner Thread erzeugt, der sich um die Verwaltung der Prozesswrapper, die damit auch den Mutex eines Prozesses halten, kümmert. Dazu wurde eine statische Variable namens „suspendResumeExecutor“ des Typs „ExecutorService“ zur Klasse „AeRemoteDebugImpl“ hinzugefügt. „AeRemoteDebugImpl“ ist eine Steuerungsinstanz für die Engine, die von Webserviceaufrufen benutzt wird. „ExecutorService“ ist seit Java 5 im Package „concurrent“ vorhanden. Hier wird er für einen „single worker thread“ genutzt, um ausschließlich in dessen Kontext den Mutex eines Prozesses zu holen und wieder freizugeben.

Darüber hinaus wird in der statischen Map „pidToWrapper“ den Prozess-IDs ihr „AeProzess-Wrapper“ zugeordnet. Beim „suspend“ wird der Wrapper in die Map eingetragen. Dadurch, dass wir den Wrapper halten, halten wir auch den Mutex und der Prozess kann nicht weiterlaufen. Beim „resume“ geben wir den Mutex für den Wrapper wieder frei, und entfernen ihn aus der Map.

Die entsprechenden Aufgaben „suspend“, „resume“ und „getDetails“ werden als entsprechende Tasks durch den ExecutorService ausgeführt. Ein Task für „getDetails“ ist daher nötig, weil ein direkter Aufruf von „getProcessDetail“ im Kontext eines anderen Thread durchgeführt werden würde, der Prozess aber blockiert wird, weil der Mutex vom ExecutorService gehalten wird. Somit käme es zu einem Timeout.

Darüber hinaus wird vor bzw. nach dem speziellen „suspend“ und „resume“ ebenfalls das „suspend“ und „resume“ der original ActiveBPEL Implementierung durchgeführt, um in der gesamten Engine korrekte Zustandsanzeigen zu garantieren, die sonst zusätzlich engineweit implementiert werden müssten, und zusätzlich von den oben beschriebenen Eigenschaften zu profitieren.

6.5 Durchführung einer Ad-hoc-Modifikation

Im Folgenden wird darauf eingegangen, wie eine Ad-hoc-Modifikation auf Seiten der Engine realisiert wird. Grundsätzlich muss die Änderung einer Prozessdefinition für eine bereits existierende Instanz als Ziel haben, das Deployment für die neue Prozessdefinition durchzuführen und dabei die betroffene Prozessinstanz so weit wie möglich wiederherzustellen. Natürlich darf dabei die geänderte Prozessdefinition nicht für alle Prozesse, die die betroffene Prozessdefinition haben, übernommen werden, sondern nur für die konkrete Instanz, die geändert wurde. Darüber hinaus muss der Engine in irgendeiner Art und Weise bekannt gemacht werden, wie sich die Prozessdefinition geändert hat. Die PG hat sich zu beiden Wegen Gedanken gemacht. Aus dem PG-Konzept entstehen immer BPEL Prozesse. Ferner beachte man im Folgenden auch, dass der Begriff Zustand als Synonym für die Kombination eines Aktivitätszustand und die Daten dieser Aktivität benutzt wird.

Erste Idee

Wenn man bedenkt, dass die Prozessdefinition als XML Dateien vorliegen, kommt man schnell auf die Idee eine Differenzanalyse („diff“) auf diese Textdateien durchzuführen um so zu erkennen, was in der Prozessdefinition hinzugekommen oder nicht mehr vorhanden ist. Wenn man diesen Ansatz weiterverfolgt, könnte man nun der geänderten Prozessdefinition die Unterschiede zur ursprünglichen Prozessdefinition beim Deployment mitgeben, oder der Engine sogar ausschließlich die Änderungen übergeben. Bei näherer Betrachtung dieser Prozedur stellt man jedoch schnell fest, dass es nicht so simpel bleiben kann: Bei zwei semantisch identischen Prozessdefinitionen können die Elemente in der XML Datei in

unterschiedlicher Reihenfolge auftauchen. Ein simples „diff“ würde in diesem Fall aber massive Änderungen der Prozessdefinition erkennen. Dies ist nur eines der Probleme, die sich aus diesem Vorgehen ergeben. Die PG wollte aber einen möglichst pragmatischen Ansatz wählen und hat sich daher für eine andere Lösung entschieden.

Pragmatismus

Folgender Ansatz für die Durchführung von Ad-hoc-Modifikationen hat sich als vielversprechend herausgestellt und wurde auch erfolgreich umgesetzt. Grundlage für die Identifizierung der Unterschiede einer Prozessdefinition ist die Dekoration sämtlicher XML Tags der Prozessdefinition mit eindeutigen, fortlaufenden und sich nicht wiederholenden IDs direkt in der BPEL Datei. Neue Elemente bekommen also auch immer eine nächsthöhere ID. Wenn Elemente zuvor gelöscht wurden, wird trotzdem bei der höchsten je für diesen Prozess vergebenen ID fortgefahren.

Diese Dekoration könnte innerhalb des Designers automatisiert mit der BPEL-Code-Generierung aus der DSL heraus stattfinden bzw. für eine Ad-hoc-Modifikation können diese IDs für den Editor festgehalten und entsprechend übernommen werden.

Momentan findet die Vergabe von IDs allerdings innerhalb des BPR-Builders statt, weil die Implementierung der Funktionalität dort einfach möglich war. Zu Testzwecken und zur Bewertung der Machbarkeit ist diese Lösung ausreichend.

Wesentlich bei dem Ansatz ist, dass die IDs als BPEL-Extension-Attributes umgesetzt werden und damit weiterhin standardkonformer BPEL-Code vorliegt, der von ActiveBPEL unterstützt und verarbeitet werden kann.

Als weiteres Attribut wird ein Flag eingeführt, das festlegt, ob für die Prozessdefinition ein spezielles Ad-hoc-Deployment durchgeführt wird. Wenn man nun ein Element der Prozessdefinition hinzufügt, in dieser Prozessdefinition das Flag setzt, ein BPR daraus baut und dafür dann in der Engine das Deployment durchführt, wird diese Ad-hoc-Prozessdefinition auch als BPR-Datei in dem BPR Verzeichnis der Engine für jeden Prozess eindeutig im Tomcat abgelegt. Der Name der Datei muss sich von der ursprünglichen Prozessdefinition unterscheiden, weil bei einem eventuellen Neustart des Server auch das Deployment sämtlicher BPR Dateien erneut durchgeführt wird. Daher müssen beide Prozessdefinitionen verfügbar sein.

Darüber hinaus werden sämtliche Deployments in der Engine verwaltet und dabei wird zur Identifizierung der Prozessname verwendet. Auch dort muss die Ad-hoc-Prozessdefinition einen eigenen Namen haben, da auch das Deployment dieser Ad-hoc-Prozessdefinition separat verwaltet werden muss.

Für die Aktivitäten des Prozesses, die auch nach einer Ad-hoc-Modifikation noch vorhanden sind, müssen die Zustände wiederhergestellt werden. Dafür hat sich die PG den Persistenzmodus der Engine zu Nutze gemacht. Es ist also erforderlich, dass die Engine im Persistenzmodus betrieben wird. Bei der Zustandswiederherstellung wird sichergestellt, dass die

Engine die betroffene Instanz in die Datenbank schreibt und nach dem Deployment der Ad-hoc-Modifikation für ihre entsprechenden Aktivitäten die Zustände aus der Datenbank wiederherstellt.

Wenn man der Engine eine neue Prozessdefinition übergibt, die ein zusätzliches Element enthält, wird durch die dafür vergebene ID festgestellt, dass es sich um eine neue Aktivität handelt. Für dieses Element wird ein Initialzustand vergeben. Alle anderen noch vorhandenen Elemente werden auch korrekt erkannt, da die IDs übereinstimmen. Für diese werden die Zustände entsprechend wiederhergestellt. Gelöschte Elemente, die dann in der geänderten Prozessdefinition nicht mehr auftauchen, werden als gelöscht erkannt. Das funktioniert deshalb, weil für die *neue* Prozessdefinition eine Zustandwiederherstellung versucht wird. In dieser taucht das gelöschte Element aber nicht mehr auf. Trivialerweise ist eine Zustandswiederherstellung in diesem Fall nicht nötig.

Bei Ersetzungen von Elementen wird eine völlig neue ID in der Prozessdefinition vergeben. Dafür wird dann durch die Migration, wie oben für ein neues Element, durchgeführt und bekommt damit auch einen Initialzustand. Man mache sich klar, dass dieses Element im Blick auf die alte Prozessdefinition auch ein neues Element ist und das alte „gelöschte“ Element nicht mehr hergestellt wird.

Grenzen

Wie bereits erwähnt findet das vorgestellte Vorgehen bei Ad-hoc-Modifikationen auf BPEL Ebene statt. Dieses Vorgehen stellt keine befriedigende Lösung sämtlicher denkbarer Ad-hoc-Fälle im BPEL-Kontext dar. Man beachte dabei, dass Ad-hoc-Modifikationen in der Domäne der PG und im Kontext des erarbeiteten Konzept jedoch auf Ebene der DSL stattfinden. Das Konzept erlaubt nur das Hinzufügen, Löschen und Ersetzen von Aktionsplänen innerhalb eines Workflows. Jeder Aktionsplan an sich stellt einen Prozess dar, für den ein Deployment in der Engine durchgeführt wurde und der in dem Workflow eines klinischen Pfades aufgerufen wird. Damit beschränken sich die Ad-hoc-Fälle im wesentlichen auf Hinzufügen, Löschen und Ersetzen (welches eine Kombination aus Löschen und Hinzufügen ist) von BPEL-Invokes der Aktionspläne innerhalb des betroffenen Workflows. Diese Fälle werden von dieser Lösung voll abgedeckt.

Nach der recht abstrakten Beschreibung der Durchführung einer Ad-hoc-Modifikation, wird im folgenden auf interessante technische Details für die Implementierung in diesem Zusammenhang eingegangen.

6.5.1 Namespace DfEngineExtensions als BPEL Extension

Wie bereits erwähnt, wurde für die Dekoration der BPEL-Dateien mit IDs und zur Kennzeichnung einer Ad-hoc-Prozessdefinition von BPEL-Erweiterungen Gebrauch gemacht. Die BPEL-Spezifikation sieht solche Erweiterungen vor und ActiveBPEL unterstützt diese auch

entsprechend. Die jeweiligen Attribute nutzen dazu den Namespace „DfEngineExtensions“. Die Verwaltung einer BPEL-Erweiterung innerhalb der Engine findet über ein Objekt der Klasse „AeExtensionElementDef“ (für die Erweiterung um ein neues Element) bzw. „AeExtensionAttributeDef“ (Erweiterung um ein neues Attribut) statt. Beide Klassen erben von „AeBaseXmlDef“. Diese Klasse repräsentiert eine allgemeine XML-Definition und mit ihr kann eine Hierarchie von Definitionen erzeugt werden, da eine Instanz der Klasse durch das Klassenattribut „mParent“ eine Referenz auf eine andere Instanz dieses Typs oder Untertyps enthalten kann. Die Klasse „AeBaseXmlDef“ stellt die öffentliche Methode „getExtensionAttributeDef(QName aAttributeName)“ zur Verfügung, die für den übergebenen QName ein AeExtensionAttributeDef zurückgibt. Sollte dieses nicht existieren, wird 'NULL' zurückgegeben.

Darüber hinaus wird eine Prozessdefinition innerhalb der Engine durch eine Instanz der Klasse „AeProcessDef“ repräsentiert. Diese erbt über mehrere ihrer Superklassen auch von „AeBaseXmlDef“. Somit kann über eine Prozessdefinition des Typs „AeProcessDef“ auch direkt ihre zugehörigen Extension Attribute und Elemente abgefragt werden. Dieses wird in der Implementierung der Ad-hoc-Modifikationen benutzt, da nahezu alle Prozessdefinitionen in ActiveBPEL über Instanzen der Klasse „AeProcessDef“ verwaltet werden.

Abbildung 6.33 veranschaulicht diesen Zusammenhang. Die Klassen enthalten alle wesentlichen Mutator- und Accessor-Methoden für ihre Attribute. Zugunsten der Übersichtlichkeit werden diese und weitere Methoden in der Abbildung nicht dargestellt.

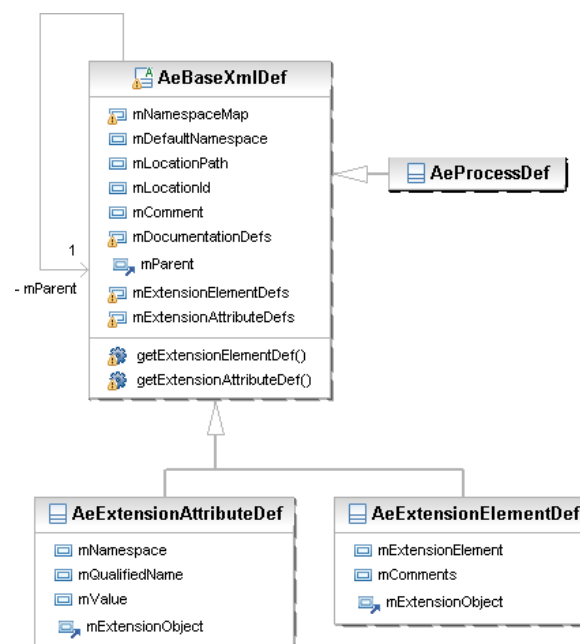


Abbildung 6.33: Zusammenhang Prozessdefinition und BPEL Extensions

In der Umsetzung resultieren daraus die folgenden BPEL-Extensions. Man beachte, dass es sich hierbei ausschließlich um zusätzliche Attribute handelt. Der Namespace wird im

Quelltext der Engine unter Umständen durch die Konstante „DF_EXTENSION_NS“ repräsentiert.

- DfEngineExtensions:adHocEnabled
Taucht DfEngineExtensions:adHocEnabled="true" auf, wird die Prozessdefinition als ad hoc- Fall identifiziert und dafür ein spezielles Deployment durchgeführt. Für alle Prozessdefinition, die dieses Flag nicht gesetzt haben, wird eine Sonderbehandlung also ausgeschlossen, und es ist demnach auch keine Ad-hoc-Modifikation solcher Prozesse möglich.
- DfEngineExtensions:maxId
Gibt die höchste für XML Tags vergebene ID innerhalb der Prozessdefinition an (z. B. DfEngineExtensions:maxId="1053").
- DfEngineExtensions:id
Stellt die für das jeweilige XML Element vergebene ID dar(z. B. DfEngineExtensions:id="1001"). Dieses Attribut wird im Quelltext der Engine unter Umständen durch die Konstante „ID_ATTRIBUTE_NAME“ repräsentiert.

Ein Beispiel für die Verwendung dieser Attribute stellt der Auszug aus einer Test-Prozessdefinition im Listing 6.11 dar. Für den Namespace „DfEngineExtensions“ ist dort das Präfix „nsdf“ benutzt worden. Dieser Prozess ist als Ad-hoc--Fall markiert (Ende Zeile 2), jedes XML-Tag ist mit einer ID gekennzeichnet. Sollten weitere IDs vergeben werden, müssen diese ab „1054“ (Ende Zeile 2) fortgeführt werden. Zeile 3 bis 5 kennzeichnen den Namespace „DfEngineExtensions“ als BPEL-Extension und wird dadurch von der Engine entsprechend verwaltet.

Listing 6.11: Auszug aus einer Test-Prozessdefinition

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <bpel:process xmlns:bpel=
   "http://docs.oasis-open.org/wsbpel/2.0/process/executable" ...
   xmlns:nsdf="DfEngineExtensions" name="Workflow"
   suppressJoinFailure="yes" targetNamespace="http://example"
   nsdf:maxId="1053" nsdf:adHocEnabled="true">
3 <bpel:extensions nsdf:id="1001">
4   <bpel:extension mustUnderstand="no"
     namespace="DfEngineExtensions" nsdf:id="1002" />
5 </bpel:extensions>
6
7 ...
8
9   <bpel:flow nsdf:id="1011">
```

```

10 <bpel:links nsdf:id="1012">
11   <bpel:link name="L1" nsdf:id="1013" />
12   ...
13 </bpel:links>
14
15 ...
16
17 <bpel:assign name="Assign666toResp" nsdf:id="1048">
18   <bpel:targets nsdf:id="1049">
19     <bpel:target linkName="L3" nsdf:id="1050" />
20     ...
21   </bpel:targets>
22 </bpel:assign>
23 </bpel:flow>
</bpel:process>

```

6.5.2 Präambel Ad-hoc-Deployment

Um die Implementierung des Ad-hoc-Deployments nachvollziehen zu können, werden nun einige wichtige Klassen und deren Beziehungen beschrieben. Abbildung 6.34 stellt diesen Sachverhalt dar. Man beachte, dass die beschriebene Klassenlandschaft in diesem Zusammenhang bei weitem nicht vollständig ist und zu Gunsten der Übersichtlichkeit viele Attribute und Methoden, insbesondere Accessor- und Mutator-Methoden, im Allgemeinen nicht dargestellt sind. Zu jedem Prozess wird das Deployment seiner jeweiligen Prozessdefinition im Speicher der Engine durch ein Objekt der Klasse „AeProcessDeployment“ verwaltet. Diese Klasse implementiert das Interface „IAeProcessDeployment“, welches wiederum vom Interface „IAeProcessPlan“ erbt. Die Vererbungshierarchie setzt sich von dort zwar über weitere Superklassen fort, soll hier jedoch nicht weiter erläutert werden. „IAeProcessPlan“ beschreibt einige grundlegende Eigenschaften eines Prozesses, die von der Engine benötigt werden, um eine Nachricht an den Prozess zu befördern. Insbesondere ist hier die Schnittstelle beschrieben, um die Prozessdefinition und eventuelle Erweiterungen abzurufen.

Das Interface „IAeProcessDeployment“ spezialisiert das Interface „IAeProcessPlan“ in der Art, dass zusätzlich die Informationen aus dem „Process Deployment Descriptor“ verwaltet werden können. Im Folgenden sind Objekte des Typs „IAeProcessDeployment“ immer Instanzen der Klasse „AeProcessDeployment“. Die Verwaltung eines Prozessdeployments in ActiveBPEL enthält auf dieser Hierarchiestufe nicht die Dateinamen der zugehörigen BPR-Datei. Diese Information wird aber von der PG zur Umsetzung der Ad-hoc-Funktionalität benötigt und wurde daher im Zuge der Implementierung ergänzt. Die Methode „getBprFileName“ liefert den Dateinamen für das Deployment als String zurück. Die zugehörige Information wird aus einem Objekt des Typs „IAeDeploymentContext“, das an den Konstruktor des Deploymentobjektes übergeben wird, mit Daten gefüllt. Diese Kontextobjekte sind Instanzen der Klassen „AeBprContext“ bzw. „AeUnpackedBprContext“, die „IAeDeploymentContext“ implementieren. Eine Unterscheidung zwischen diesen Klassen wird hier nicht gegeben und es wird auch nicht detaillierter auf die Verflechtung dieser Klassen in den

Standard-Deploymentvorgang von ActiveBPEL eingegangen, da diese für das Verständnis der Implementierung nicht weiter benötigt wird und die Problematik des PG-Temas verlässt.

Man beachte, dass die in [6.5.1](#) beschriebene Klasse eine Referenz auf das zugehörige Deploymentobjekt enthält.

Die Verwaltung sämtlicher Deployments aller Prozessdefinitionen findet in der Klasse „AeDeploymentProvider“ statt, die von „AeAbstractDeploymentProvider“ erbt und so das entsprechende Interface „IAeDeploymentProvider“ implementiert. Diese Klasse stellt Methoden zur Verfügung, um Deployments aufzunehmen, zu entfernen, Iteratoren für die Deployment-HashMap zu liefern und Deployments für einen übergebenen QName eines Prozesses zu liefern. Auch hier wurde eine Methode hinzugefügt, die direkt den Dateinamen des zugehörigen Deployments für einen übergebenen „QName“ eines Prozesses liefert.

„IAeMessageContext“ liefert der Engine Kontextinformationen zu einer eingehenden Nachricht (z. B. welcher Prozess in einem „invoke“ benutzt wird). „IAeMessageContext“ wird in „IAeProcessDeployment“ als Typ, z. B. für Methodenparameter, benutzt. Daher besteht zwischen diesen Klassen eine Abhängigkeit. In der Implementierung des Interface gleicht die Engine über diese Informationen z. B. „Partnerlinks“ und entsprechende „Endpoint References“ ab.

Darüber hinaus erweitert die Klasse „AeExtendedMessageContext“ die Klasse „AeMessageContext“ um die zusätzlich benötigten Informationen, um SOAP Nachrichten abzuarbeiten.

6.5.3 Ad-hoc-Deployment

Wenn an die Engine eine Anfrage gestellt wird, einen neuen Prozess zu erzeugen, führt die Engine verschiedene Vorgänge aus. Im Laufe dieser Arbeiten wird unter anderem in der Klasse „AeAbstractReceiveHandler“ die Methode „handleReceiveData“ aufgerufen. Dort setzt die Implementierung der PG an, um ein Ad-hoc--spezifisches Deployment sicherzustellen. Die Klasse „AeAbstractReceiveHandler“ implementiert die Interfaces „IAeReceiveHandler“ und „IAeBPELReceiveHandler“ und stellt damit die Funktionalität zur Verfügung, um „Receives“ zu empfangen und zu verarbeiten.

Listing [6.12](#) stellt die Methode „handleReceiveData“ dar. Beim Aufruf werden die Parameter „aData“ vom Typ „IAeWebServiceMessageData“ und „aContext“ vom Typ „IAeMessageContext“ übergeben. In „aContext“ sind die Informationen der Anfrage enthalten, welcher Prozess betroffen ist. Abhängig davon wird der entsprechende Prozessplan und damit die zugehörige BPR-Datei geladen. Man beachte in diesem Zusammenhang die Ausführungen zur Klassenlandschaft in Kapitel [6.5.2](#).

Für einen Ad-hoc-Fall kann man nun aber nicht einfach diese Prozessdefinition ändern, sondern benötigt eine Prozessdefinition, die nur für die betroffene Instanz gilt. Sonst würden die Änderungen alle Instanzen dieser Prozessdefinition betreffen. Daher wird in diesem Fall sichergestellt, dass eine für die Instanz exklusive Prozessdefinition existiert. Ebenso muss

der Prozessname eindeutig sein.

In Zeile 28 wird daher zunächst die Prozessdefinition des betroffenen Prozesses aus dem Nachrichtenkontext geladen. Daraus wird in Zeile 30 dann die Prozessdefinition geladen und diese auf das BPEL-Extension-Attribut „adHocEnabled“ untersucht. Wird dieses gefunden, wird in Zeile 33 der Wert dafür ausgelesen. Näheres zur Nutzung der BPEL-Extension im PG-Konzept findet man in Kapitel 6.5.1.

Wenn es sich um eine ad hoc markierte Prozessdefinition handelt, wird für diese das benötigte exklusive Deployment in Zeile 39 sichergestellt. Danach läuft die Methode mit den Anweisungen weiter, die in ActiveBPEL standardmäßig vorhanden sind.

Es sei hier erwähnt, dass der Aufruf von „createUniqueContext“ in Zeile 39 auch von der PG implementiert wurde und einige komplexe Operationen durchführt. Auf die technischen Details dieser Implementierung wird hier nicht weiter eingegangen, da diese für das PG-Thema und die gefundene Umsetzung nicht weiter interessant sind. Die Implementierung beinhaltet im Wesentlichen die Erzeugung eines eindeutigen Prozessnamens und die eindeutige Bezeichnung für die BPR Datei.

Listing 6.12: Methode handleReceiveData der Klasse AeAbstractReceiveHandler

```

24  public IAeWebServiceResponse
    handleReceiveData (IAeWebServiceMessageData aData ,
    IAeMessageContext aContext) throws
    AeBusinessProcessException {
25  AeExtendedMessageContext context =
    AeExtendedMessageContext .convertToExtended (aContext);
26
27  boolean isAdHocEnabledProcess=false ;
28  final IAeProcessDeployment orgDeployment =
    AeProcessDeploymentFactory .getDeploymentForPlan(
    getProcessPlan (context));
29
30  AeExtensionAttributeDef
    adHocEnabledAttribute=orgDeployment .getProcessDef ()
    .getExtensionAttributeDef (new QName(
    DfActivityStateMappingVisitor .DF_EXTENSION_NS
    , "adHocEnabled"));
31  if (adHocEnabledAttribute != null) {
32  try {
33  isAdHocEnabledProcess=Boolean .valueOf(
    adHocEnabledAttribute .getValue ());
34  } catch (Exception e) {
35  }
36  }
37
38  if (isAdHocEnabledProcess) {
39  context=createUniqueContext (context);
40  }

```

```
41     |AeProcessPlan plan = getProcessPlan(context);
42     updateContextWithPlanInfo(plan, context);
43     authorizeRequest(plan, context);
44     validateInputData(plan, context, aData);
45     |AeMessageData data = mapInputData(plan, context, aData);
46     return invokeProcessEngine(plan, context, data);
47
48 }
```

6.5.4 Prozessinstanzmigration

Wie bereits in der Einführung dieses Kapitels erläutert, wird der Persistenzmechanismus der Engine genutzt, um eine Prozessinstanz auf eine neue Prozessdefinition zu migrieren. Dabei wurde beim Anhalten des Prozesses sichergestellt, dass dieser persistent in der Datenbank vorliegt und nicht mehr im Speicher der Engine verfügbar ist. Eine exklusive Prozessdefinition liegt nach den Ausführungen in 6.5.3 für den Prozess ebenfalls vor. Nun wurde diese geändert und das Deployment wird erneut durchgeführt. Der Prozess wird nun mit der Aktion „resume“ fortgesetzt. Dabei holt die Engine den Prozess aus der Datenbank und macht ihn damit wieder im Speicher verfügbar. Bei diesem Vorgang greift die Implementierung der PG ein und sorgt für die Migration der Instanz in die neue Prozessdefinition.

Die entsprechenden Codepassagen verteilen sich über die Klassen „AeLocationPathVisitor“, „AeRestoreImplState“ und „AeProcessImplState“.

„AeProcessImplState“ sorgt für das Speichern der Zustände eines Prozesses. Diese Klasse wurde von der PG um die Methode „serializeBpelObject“ erweitert. Dieser Methode wird ein BPEL-Objekt des Typs „AeAbstractBpelObject“ als Parameter übergeben. Für dieses Objekt erzeugt die Methode ein XML-Dokument mit den zugehörigen Statusinformationen und gibt dieses als XML-Element zurück.

Die Objektstruktur der einzelnen BPEL Elemente wurde in ActiveBPEL über das Visitorpattern realisiert. Die Klasse „AeLocationPathVisitor“ besucht dementsprechend dieses Objekt-Modell und weist jeder Definition einen entsprechenden „location path“ zu. Zu jedem „location path“ wird auch eine „location id“ vergeben. Die Beziehung zwischen ihnen wird in einer Java-Map verwaltet. Ursprünglich wurden diese IDs fortlaufend hochgezählt. Es wurde durch die PG-Implementierung aber nun die Möglichkeit geschaffen, dass diese IDs bei einer geänderten Prozessdefinition für die einzelnen Elemente bei einem erneuten hochzählen nicht mit der neuen Prozessdefinition konsistent sind. Daher werden nun hier die IDs benutzt, mit der die BPEL-Elemente dekoriert wurden. Dazu musste die Methode „updateLocationId“ in der Klasse „AeLocationPathVisitor“ angepasst werden, die in Listing 6.13 dargestellt ist. Der Methode wird das XML-Element übergeben, für das die „location id“ angepasst werden soll. Aus diesem wird, wie in Kapitel 6.5.1 beschrieben, die ID der

Dekoration ausgelesen und als „location id“ benutzt (Zeile 51). Wenn dieses Extension Attribut nicht verfügbar ist, handelt es sich auch nicht um ein BPEL-Element. Daher kann für diese Elemente nach dem ursprünglichen Verfahren die entsprechende fortlaufende ID vergeben werden (Zeile 57 und 58). Anschliessend wird die ID für den „location path“ zwecks Verwaltung in die „LocationPathMap“ hinzugefügt.

Es sei hier erwähnt, dass die Dekoration der BPEL Elemente bei 1000 startet, weil die „location ids“ anderer, nicht „BPEL-Def“ Elemente ebenfalls hier stattfindet. Wir gehen also davon aus, dass der Vorrat an IDs für die anderen Elemente ausreicht, diese also von der Anzahl her nicht 1000 erreichen. Sollte das nicht ausreichen, kann die Start-ID für die Dekoration natürlich auch höher angesetzt werden. Natürlich wäre ein vollständig eigenes Attribut für diese IDs innerhalb der Engine wünschenswert. Der zeitliche Implementationsaufwand dafür hat sich schnell als recht hoch herausgestellt. Daher hat sich das beschriebene Vorgehen durchgesetzt, den bereits vorhandenen „Location IDs“ die ID-Werte der BPEL-Prozessdefinition aufzuzwingen. Eine Trennung der IDs innerhalb der Engine, indem die IDs der BPEL-Prozessdefinition z. B. „MeinID21“ lauten, wäre aufgrund des Datentyps Integer für dieses Attribut innerhalb der Engine auch nur durch erhöhten Aufwand möglich gewesen.

Listing 6.13: Methode updateLocationId der Klasse AeLocationPathVisitor

```

49 protected void updateLocationId(AeBaseXmlDef aDef)
50 {
51     final AeExtensionAttributeDef
52         extAttribute=aDef.getExtensionAttributeDef(new
53             QName("DfEngineExtensions","id"));
54     final int locationId;
55
56     if (extAttribute != null) {
57         locationId=Integer.parseInt(extAttribute.getValue());
58     } else {
59         locationId = getNextLocationId();
60         setNextLocationId(locationId + 1);
61     }
62
63     getLocationPathMap().put(getPath(), locationId);
64     recordLocationPathAndId(aDef, getPath(), locationId);
65 }

```

Die Klasse „AeRestoreImplState“ sorgt für die Wiederherstellung der „BPEL-Implementation“ Objekte, die über die Klasse „AeProcessImplState“ gespeichert wurden. Wenn beim Fortsetzen des Prozesses die Zustände wiederhergestellt werden, wird von der Methode „getElement“ für ein übergebenes „Implementations“-Objekt das entsprechende XML-Element zurückgegeben, das dann auch wieder den ursprünglichen Zustand enthält. Die Methode ist in Listing 6.14 dargestellt.

Es wird als Erstes das Element für den „location path“ geholt (Zeile 68). Man beachte, dass „getElement“ in dieser Klasse überladen ist. Die Methode mit der hier in Zeile 68

aufgerufenen Signatur wird in Zeile 87 bis 97 dargestellt. Diese Methode gibt das Element zurück, das den Zustand enthält, der durch den „location path“ des übergebenen Objektes beschrieben ist.

Falls dieser Aufruf eine Exception wirft, gibt es keinen gespeicherten Zustand für das Element. Daher wird die Exception zwar gefangen, jedoch nicht weiter behandelt (Zeile 69 bis 70). Das ist an der Stelle sinnvoll, weil für ein neues Element, das durch eine Ad-hoc-Modifikation in dem Prozess hinzugefügt wurde, kein entsprechendes Element mit einem Zustand existieren kann.

Wenn es kein Element gab und es sich um ein BPEL-Objekt handelt, wird für dieses Element mit dem Aufruf der oben beschriebenen Methode „serializeBpelObject“ ein entsprechendes XML-Element erzeugt und anschließend auf den Zustand „QUEUED_BY_PARENT“ gesetzt (Zeile 72 bis 83). Dieses wird explizit gesetzt, weil der Initialzustand eines BPEL-Objektes „INACTIVE“ ist. Wir tun dies, weil wir davon ausgehen, dass wir uns in einem großen „Flow“ befinden und dürfen daher alle Objekte auf den Zustand „QUEUED_BY_PARENT“ setzen. Ausgeführt werden sie erst dann, wenn ihre „links“ ausgewertet wurden. Würde man hier nicht so verfahren, könnten neue Elemente nie ausgeführt werden. Anschließend wird das Ergebnis zurückgegeben.

Wenn der Aufruf in Zeile 68 erfolgreich war, liegt direkt ein Element vor, das den ursprünglichen Zustand enthält. Der „if“-Block in Zeile 72 bis 83 wird nicht ausgeführt und es wird das gefundene Element zurückgegeben.

Listing 6.14: Methode getElement der Klasse AeRestoreImplState

```

64  public Element getElement(IAeLocatableObject almpl) throws
        AeBusinessProcessException
65  {
66      Element element=null;
67      try {
68          element = getElement(almpl.getLocationPath());
69      } catch (AeBusinessProcessException e) {
70      }
71
72      if(element==null){
73          if(almpl instanceof AeAbstractBpelObject){
74              AeProcessImplState state=new AeProcessImplState();
75              try {
76                  element =
77                      state.serializeBpelObject((AeAbstractBpelObject) almpl);
78              } catch (AeBusinessProcessException e) {
79                  e.printStackTrace();
80              }
81
82              element.setAttribute(STATE_STATE,
83                  AeBpelState.QUEUED_BY_PARENT.toString());
84      }
85  }

```

```
84     return element;  
85 }  
86  
87 protected Element getElement(String aLocationPath) throws  
88     AeBusinessProcessException  
89 {  
90     Element element = (Element)  
91         getLocationPathsMap().get(aLocationPath);  
92  
93     if (element == null)  
94     {  
95         throw new  
96             AeBusinessProcessException(AeMessages.getString(  
97                 "AeRestoreImplState.ERROR_0" + aLocationPath));  
98     }  
99  
100    return element;  
101 }
```



Abbildung 6.34: Überblick Klassenstruktur für das Ad-hoc-Deployment.

7 Lessons Learned

Während der Durchführung der Projektgruppe kam es an verschiedenen Stellen zu unterschiedlichen Problemen und Abweichungen vom geplanten Ablauf. Einige dieser Schwierigkeiten sorgten dafür, dass nicht alle Tätigkeiten in der vorgesehenen Zeit bewältigt werden konnten. In den nachfolgenden Abschnitten sollen die technischen und organisatorischen Ursachen der Probleme betrachtet werden.

7.1 Technische Probleme

In diesem Bereich werden Probleme betrachtet und vorgestellt, die auf technische Ursachen zurückzuführen sind. Dabei werden die einzelnen Problembereiche kurz umschrieben.

Technischer Editor

Bei der Erweiterung des technischen Editors musste zunächst der vorhandene Quelltext analysiert werden. Nachdem zu Beginn der Arbeiten schnell die ersten Erfolge erzielt wurden, verzögerte die Ergänzung des Editors um eigene BPEL-Elemente die Umsetzung von geplanten Änderungen. Ursache für die Probleme in diesem Bereich war, dass der Editor intern wesentlich komplexer aufgebaut ist, als zunächst erkennbar war. Erschwerend kam hinzu, dass der bestehende Quelltext an zahlreichen Stellen noch nicht vollständig implementiert wurde. Erkennbar war dies durch Kommentare im Quelltext und Fehler, die bei der Verwendung auftraten.

Repository

Nach der Auswahl des Repositories musste über die mitgelieferte API auf die Funktionen der Anwendung zugegriffen werden. Im Vorfeld war bekannt, dass es sich beim Repository vielmehr um eine Machbarkeitsstudie als um ein ausgereiftes Produkt handelt. Dennoch war die Einarbeitung sehr aufwändig, da die Dokumentation stellenweise große Lücken aufweist, so dass die Verwendung der bereitgestellten Funktionen mühselig analysiert werden musste. Besonders zeitaufwändig war die Analyse der Funktionen zu Beginn der Umsetzungsphase, da der Quelltext zunächst nicht zur Verfügung stand.

Prozesslisten PlugIn

Nachdem anfangs nicht bekannt war, wie das Eclipse-PlugIn aufgebaut werden muss, war das Finden einer Konfiguration mit den benötigten Bibliotheken sehr aufwändig. Die abhängigen Bibliotheken wurden zunächst mit Maven in das Projekt integriert. Allein die Konfiguration für Maven war nicht besonders einfach. Nachdem Maven richtig konfiguriert war, konnte mit der Entwicklung weitergemacht werden. Bei der Ausführung des Plug-Ins in anderen Eclipse Installationen stellte sich bald heraus, dass die importierten Bibliotheken nicht mit exportiert und das Plug-In somit nicht lauffähig war. Es folgte eine aufwändige Fehlersuche, weshalb die Bibliotheken, obwohl sie vorhanden waren, nicht vom Classloader auf anderen Systemen gefunden werden konnten. Die Entwicklung unter Windows Vista zeigte zudem, dass besonders unter diesem Betriebssystem nicht ohne weiteres auf das Dateisystem zugegriffen werden konnte. Ein ähnliches Problem zeigt sich außerdem in Linux VM, so dass beim Zugriff auf das Dateisystem die Rechte des aktuell angemeldeten Benutzers berücksichtigt werden mussten.

Fachlicher Editor

Der fachliche Editor sollte ebenfalls als Eclipse Plug-In erstellt werden. Das DynamicFlow-Metamodell erforderte eine komplette Neuentwicklung eines graphischen Editors für Workflows, da es offensichtlich keinen existierenden Editor für das Metamodell der Projektgruppe geben konnte. Eine solche Entwicklung kann aufgrund der Komplexität im Rahmen einer Projektgruppe nur funktionieren, wenn entweder ein bestehendes Produkt angepasst wird, oder der Großteil des Codes auf einfache Weise generiert werden kann.

Die Suche nach einem anzupassenden Produkt verlief ergebnislos. Somit war der einzig noch gangbare Weg ein Code-Generator. Mit den Eclipse Frameworks EMF¹, GEF² und GMF³ wurden vielversprechende Kandidaten gefunden.

Die drei Frameworks bauen aufeinander auf und erlauben eine modellgetriebene Softwareentwicklung. Mit EMF kann aus einem sogenannten *ECore* Modell der Quelltext für ein Domänenmodell und einen einfachen graphischen Editor generiert werden. GEF unterstützt die Erstellung graphischer Editoren, die das durch EMF generierte Modell bearbeiten können. GMF bietet eine Unterstützung für die Editor-Erstellung und setzt auf GEF und EMF auf.

Die Kombination dieser drei Werkzeuge erlaubte einen schnellen Anfangserfolg, der Hoffnung auf einen funktionstüchtigen Editor weckte.

Die hohe Informationsdichte des generierten EMF Codes, gekoppelt mit einer absolut mangelhaften Dokumentation haben die Verwendung von EMF allerdings erschwert. Selbst unter Zuhilfenahme des offiziellen EMF Buches[45] war der Einarbeitungsaufwand sehr hoch. Die

¹Eclipse Modelling Framework <http://www.eclipse.org/emf/>

²Eclipse Graphical Editing Framework <http://www.eclipse.org/gef/>

³Graphical Modelling Framework <http://www.eclipse.org/gmf/>

Dokumentationen für die eingesetzten Versionen der GEF/GMF Frameworks war entweder nicht vorhanden, veraltet oder schlicht unverständlich.

Die im nächsten Abschnitt besprochenen Probleme mit der Integration trafen für EM-F/GMF/GEF verschärft zu. Die Einschränkung auf Eclipse 3.3 nötigte zum Einsatz alter Versionen der Frameworks. Die verfügbaren Versionen waren noch schlechter dokumentiert, als die aktuellen Versionen und sind im Ergebnis nicht geeignet gewesen die Entwicklung eines Editors zu unterstützen. Für zukünftige Projekte kann man diese Frameworks wahrscheinlich einsetzen, dabei ist jedoch die frühzeitige Entwicklung von Expertenwissen von Nöten.

Integration

Bei dem ersten Integrationstest, also dem Betrieb der einzelnen Komponenten, zeigten sich weitere Probleme. Ein besonders schwerwiegendes Problem war, dass für die Komponenten zunächst bestimmt werden musste welche Eclipse Plug-Ins zum Betrieb der Plug-Ins vorhanden sein müssen. Dieser wichtige Schritt wurde bei der Entwicklung und der ersten in Betriebnahme der Erweiterungen versäumt, so dass die Abhängigkeiten von verschiedenen Personen mehrmals ermittelt werden mussten.

Erschwerend kam hinzu, dass die durch alle Plug-Ins entstandenen Abhängigkeiten Probleme bereiteten. Besonders die Abhängigkeit von der veralteten Eclipse Version 3.3 führte zu dem Effekt, dass eine Installation und Wartung der notwendigen Komponenten über den Eclipse-Update Manager nicht möglich war. Es gab in der Summe immer unauflösbare Konflikte, da zueinander inkompatible Pluginversionen benötigt wurden.

In einem längeren Prozess des Versuch & Irrtums konnte schliesslich eine funktionierende Eclipse Installation erstellt werden. Der dafür beschriebene Weg funktionierte nach zwei Montaten schon nicht mehr reproduzierbar.

Diese Reibungsverluste führten nicht nur zu vielen technischen Problemen, sie schlug vor allem sehr auf die Motivation. Anstatt „produktiv zu arbeiten“ musste man sich fast täglich mit sich aus den Plug-In Abhängigkeiten resultierenden Problemen beschäftigen. Dieser konstante Frust hat die Entwicklung der graphischen Komponenten deutlich gehemmt.

Kurz vor Ende der PG ist es gelungen eine (halbwegs) funktionierende Arbeitsumgebung aufzusetzen und alle darin vorhandenen Plug-Ins über eine eigene Eclipse Update-Site zur Verfügung zu stellen. Leider ist diese Lösungsmöglichkeit erst nach dem eigentlichen Entwicklungsende entdeckt worden. Für weiterführende Projekte kann nur dringend dazu geraten werden, eine solche Update-Site einzurichten und zu pflegen. Nur so ist eine reproduzierbare Build-Umgebung zu erstellen. In wie weit dieser Ansatz in der Praxis umsetzbar ist, muss sich allerdings noch zeigen.

7.2 Einstieg in ActiveBPEL mit AspectJ

Als Team C mit der Sichtung des ActiveBPEL Quelltext begann, waren alle Beteiligten zuerst vom Umfang und der Komplexität der Engine überrascht. Im Quelltext wird intensiv mit Interfaces und Vererbung dieser Interfaces gearbeitet, so dass bei einer reinen Code Sichtung nicht klar ist, welche Klassen tatsächlich zur Laufzeit ineinander greifen. Dies lässt sich zwar durch Debugging des Servers zur Laufzeit feststellen, für einen allgemeinen Überblick über die Engine ist dieses Vorgehen allerdings zu aufwändig. Um dennoch zügig einen Überblick über die Abläufe und internen Vorgänge der Engine im Betrieb zu bekommen, wurden Möglichkeiten gesucht einen schnellen Einstieg zu finden. Als Schlagwort hatte man bereits schon öfter von aspektorientierter Programmierung gehört, aber keiner hatte sich bereits im Detail damit beschäftigt. Also war das eine gute Gelegenheit dies nachzuholen.

Ziel war es z. B. festzustellen, wann bestimmte Methoden in welcher Reihenfolge und, eigentlich war das am wichtigsten, in welchem Kontext aufgerufen werden. Dafür hat sich eine spezielle Logging-Funktionalität angeboten, die mit aspektorientierter Programmierung schnell und ohne Eingriffe in den existierenden Code realisieren lässt.

Logging oder auch Tracing sind typische Beispiele sogenannter „cross cutting concerns“, dies sind Funktionalitäten, die nicht durch eine einzelne Klasse im Sinne der objektorientierten Programmierung gekapselt werden können, sondern in der Regel mehrere Klassen oder gleich mehrere Projekte betreffen. Mit Hilfe der aspektorientierten Programmierung, lässt sich dennoch ein „separation of concerns“ erreichen, indem der querschneidende Belang „Logging“ als einzelner Aspekt implementiert wird. Damit ist ein Ziel der aspektorientierten Programmierung, die Modularität und Wartbarkeit von Softwareprojekten über das objektorientierte Programmierparadigma hinaus zu verbessern. Im Folgenden wird berichtet, welche Tools in diesem Zusammenhang benutzt wurden und eine kurze Einführung in AspectJ gegeben. Die PG bewegte sich im JAVA Umfeld. Um also von einer Aspektorientierung profitieren zu können, musste eine Spracherweiterung von JAVA benutzt werden. Die PG hat sich für AspectJ[82] entschieden. Für AspectJ gibt es ein Toolkit für Eclipse namens AJDT (AspectJ Development Toolkit)[94]. In diesem Toolkit ist bereits AspectJ enthalten. Da alle Beteiligten mit Eclipse entwickelt haben, war also nur AJDT zu installieren.

Die wesentlichen Sprachkonstrukte von AspectJ sind:

- JoinPoint
Ein Joinpoint ist ein Punkt in dem Ausführungsfluss des zugrundeliegenden Systems
- PointCut
Ein Pointcut ist Abfrageprädikat über Joinpoints

- Advice
Ein Advice ist der Code, der ausgeführt werden soll wenn ein JoinPoint erreicht wurde, der einen Pointcut erfüllt. Dieser Code wird dann vor, nach oder anstatt des JoinPoint ausgeführt.

- Aspect
Ein Aspect kapselt die Definition von Advices und Pointcuts

Joinpoint

„Public static void main(String[]args)“ ist schon ein Joinpoint. Dieser ist aber ziemlich speziell, denn normalerweise möchte man z. B. mehrere Methodenaufrufe ansprechen. Man kann dazu auch Wildcards einsetzen: `Public * MeineKlassexy.set*(..)`
Dieser Pointcut bezeichnet alle öffentlichen Methoden mit einem beliebigen Rückgabewert in der Klasse `MeineKlassexy`, bei denen der Methodename mit `set` beginnt und die beliebige Parameter haben.

Pointcut

Ein Pointcut ist eine Sammlung beliebig vieler Joinpoints, die als Ganzes über den Namen des Pointcut ansprechbar sind:

```
pointcut greeting():  
execution(* HelloWorld.sayHello(..));
```

Es wird die Sammlung der Joinpoints, die hier nur aus dem einen Joinpoint `execution(* HelloWorld.sayHello(..)` besteht, mit dem Namen `greeting` versehen.

Advice

Der Advice bestimmt, welches Vorgehen für welche Pointcuts angewendet wird.

```
after() returning:greeting()  
System.out.println("World!");
```

Nach der Rückgabe von einem Joinpoint aus dem Pointcut `greeting` gibt dieser Advice „World!“ auf der Konsole aus.

Es gibt folgende Arten von Advices:

- **Before**
Direkt vor Ausführung des Pointcuts
- **After**
Nach return oder throwing des Pointcuts
- **Around**
Kontrolliert beim Erreichen des Joinpoints, die weitere Ausführung des Joinpoints

Aspect

Die Sammlung von Pointcuts und anderem Code wird in einem Aspect gekapselt. Listing 7.1 stellt einen Aspect dar.

Listing 7.1: Ein Aspect

```
98 public aspect World {
99
100     pointcut greeting():
101         execution(* HelloWorld.sayHello(..));
102
103     after() returning : greeting() {
104         System.out.println("World!");
105     }
106 }
```

AJDT

AJDT ermöglicht eine komfortable Aspektverwaltung mit visualisierenden Plugins, die z. B. direkt im Code zur Anwendung kommende Aspekte graphisch anzeigen und man direkt in den Code des Aspektes springen kann. Da Aspekte eventuell Auswirkungen haben, die im normalen Code des Basissystems nicht ersichtlich sind, empfiehlt es sich in jedem Fall von einem Toolkit wie AJDT Gebrauch zu machen, obwohl die Aspektorientierung auch allein mit einer reinen Spracherweiterung wie ApectJ umsetzbar wäre.

7.3 Organisatorisch

Wie zuvor im technischen Bereich, werden an dieser Stelle die Probleme aufgeführt, die aus organisatorischen Gründen aufgetreten sind.

Absprachen

In kleineren Gruppen wurden eine Vielzahl von Vorgängen besprochen, diskutiert und letztendlich beschlossen. Die Entscheidungsfindung wurde in einigen Fällen dabei leider nicht mit dokumentiert, so dass einige Aspekte erneut besprochen wurden. In Fällen, in denen die Informationen zur Verfügung standen, zeigte sich jedoch auch, dass nicht alle Teammitglieder über den aktuellen Stand des Gesamtprojekts informiert waren. Es wurde schnell deutlich, dass auch Informationen zu Modulen, an denen man nicht direkt beteiligt war, aufgenommen werden mussten. Folglich kam es jedoch dazu, dass bei vielen nur ein Teil des Gesamtwissens der PG vorhanden war. Es wäre hilfreich gewesen wenn alle Mitglieder einen groben Überblick über die Gesamtzusammenhänge der Module gehabt hätten.

Planung

Im Bereich der Planung zeigte sich, dass es sinnvoller war konkrete Termine für einzelne Tätigkeiten zu definieren, da nur so eine Nachverfolgung des Fortschritts möglich ist. In Fällen wurden Termine jedoch nicht eingehalten, beispielsweise aufgrund vielvältiger technischer Probleme oder es zeigte sich aber, dass die Schätzungen der benötigten Zeit aufgrund mangelnder Erfahrung viel zu optimistisch waren. In letzteren Fällen hätten auch teilweise eher andere Mitglieder des Projektteams um Unterstützung gebeten werden müssen, damit die zur Verfügung stehende Zeit besser hätte genutzt werden können.

8 Fazit und Ausblick

Die letzten Monate waren eine sehr spannende Zeit. Nicht nur fachlich und technisch, sondern vor allem auch menschlich haben alle Teilnehmer der Projektgruppe eine Menge lernen können. Alles in allem war diese Projektgruppe eine Bereicherung für unser Studium.

Das Ende des Berichts ist ein guter Platz um Bilanz zu ziehen. Was waren die Anforderungen an die Projektgruppe, was hat die Gruppe geleistet und wie ist diese Leistung zu bewerten?

8.1 Bewertung Aktueller Stand

Am Anfang des ersten Semesters wurden „Wunschziele“ an die Projektgruppe gestellt

1. Die Einarbeitung in die medizinischen wie technischen Fachlichkeiten
2. Die Auswahl und Anpassung eines bestehenden Metamodells
3. Eine explorative Implementierung eines fachlichen Editors, vorzugsweise durch das Erweitern eines existierenden Editors
4. Eine explorative Implementierung eines technischen Editors, vorzugsweise durch das Erweitern eines existierenden Editors
5. Die Anpassung einer bestehenden Workflow Engine
6. Das Adaptieren von ein oder zwei klinischen Pfaden auf das zu erstellende Metamodell

Vergleicht man die Vision vom Anfang der Projektgruppe mit dem Erreichten, so hat die Projektgruppe nicht alle diese Ziele erreichen können. Stellt man das Ergebnis neben die Ziele, so findet man bei der Implementierung des fachlichen Editors Lücken:

1. Die Einarbeitung in die medizinischen wie technischen Fachlichkeiten ist geschehen. Dieses Ziel kann als *erfüllt* angesehen werden.
2. die Auswahl und Anpassung eines bestehenden Metamodells: Es konnte kein bestehendes Modell angepasst werden, da alle untersuchten Ansätze für die Ziele der Projektgruppe unzureichend waren. Die Projektgruppe hat das geforderte neu Modell erstellt. Dieses Ziel kann ebenfalls als *erfüllt* angesehen werden.

3. Eine explorative Implementierung eines fachlichen Editors, vorzugsweise durch das Erweitern eines existierenden Editors. Es wurde zwar ein fachlicher Editor implementiert, allerdings wurde nur ein kleiner Teil der Modellierung umgesetzt. Die Integration mit den restlichen Komponenten ist nicht erfolgt.
Dieses Ziel können wir nur als *teilweise erfüllt* betrachten.
4. Eine explorative Implementierung eines technischen Editors, vorzugsweise durch das Erweitern eines existierenden Editors. Die Erstellung des technischen Editors ist gelungen. Jedoch fehlt auch hier die Integration mit dem fachlichen Editor.
In Summe kann dieses Ziel aber als *erfüllt* angesehen werden.
5. Die Anpassung einer bestehenden Workflow Engine.
Dieses Ziel kann als *erfüllt* angesehen werden.
6. Das Adaptieren eines oder mehrerer klinischer Pfade auf das neu zu erstellende Metamodell. Der Anfang des zweiten Semesters durch das ISST gelieferte Beispielpfad hat sich als gute Grundlage für die sich im Anhang befindliche Umsetzung herausgestellt. Von daher kann dieses Ziel ebenfalls als *erfüllt* angesehen werden.

Auch wenn die Implementierung der Editoren und die Umsetzung der Pfade keine zwingenden Erfolgskriterien waren, waren sie doch zentrale Anforderungen an die Gruppe. Man muss sich die Frage stellen, wo die Ursachen liegen.

Ursachenforschung

Es vor allem ein Punkt, der die Gruppe von einer technisch weiter fortgeschrittenen Implementierung getrennt hat: Es gab kein anzupassendes Metamodell. Die Planung der Projektgruppe basierte auf der Annahme, dass ein bestehendes Modell angepasst werden kann. Daraus folgte, dass auch die Editoren durch Anpassungen bestehender Produkte erzeugt werden können. Unter diesen Prämissen war das Ziel der Projektgruppe gut vollständig erreichbar.

Nach der Binsenweisheit, nach der der Beweis der Nicht-Existenz eines passenden Modells schwieriger als der Beweis der Existenz eines solchen ist, wurde die Arbeit der Gruppe nun deutlich erschwert. Die Erstellung eines eigenen Metamodells erwies sich dann auch als schwieriger, als anfangs gedacht. Nicht nur, dass die Gruppe durch diese ungeplanten Aufgaben aufgehalten wurden, es fielen mit dem neuen Modell auch die Vorteile einer Adaption weg. Für die Implementierung bedeutete das: neu statt anpassen.

Von dieser Warte aus betrachtet kann das Ergebnis der Projektgruppe *DynamicFlow* als voller Erfolg bezeichnet werden.

8.2 Ausblick und Konzeptionelle Erweiterungen

Die Projektgruppe hat nicht nur ein unter den personellen und zeitlichen Einschränkungen einer PG umsetzbares Workflowmodell gefunden, es wurden auch große Fortschritte bei der Umsetzung eines graphischen Editors und der Anbindung und Anpassung einer Workflow Engine gemacht.

Das WfM ist in seiner aktuellen Fassung darauf ausgelegt, mit möglichst wenigen verschiedenen Elementen und Operationen auszukommen. Es werden sich in der weiteren Forschung aller Wahrscheinlichkeit nach noch nützliche Erweiterungen finden. Das bisher noch sehr unterentwickelte Datenmodell könnte beispielsweise durch eine Ontologie erweitert werden. Skripte in Aktionsplänen oder gar geskriptete Modelländerungen lassen interessante Möglichkeiten erahnen.

Die folgenden Punkte stellen unseres Erachtens lohnenswerte Forschungsprojekte dar.

Zoomen

Nicht immer ist der höchste Detailgrad auch der passende. In vielen Fällen soll ein Workflow vereinfacht dargestellt werden.

Graphmorphismen und hierarchische Graphen können hier helfen, einen Workflow in verschiedenen Auflösungen zu betrachten. Sollen solche *zoom*-Möglichkeiten konzipiert werden, so ist neben der mathematischen Umsetzung vor allem die Benutzeroberfläche von höchstem Interesse. Mit der Oberfläche müssen die Grenzen zwischen Teilgraphen in intuitiver Weise durch den Benutzer definiert werden können. Die Darstellung von hierarchischen Graphen wird ebenfalls eine spannende Herausforderung.

Flexibilität und Korrektheit

Im Gegensatz zu anderen Workflow-Systemen wie ADEPT([92]), verfolgte die Projektgruppe besonders im ersten Semester den Grundsatz „Flexibilität über Korrektheit“. Dieser grundsätzliche Unterschied zu traditionellen, auf absolute Korrektheit achtenden Modelle, eröffnet spannende Möglichkeiten und eine Flexibilität in Workflows, wie sie sonst kaum möglich ist. Wir halten eine Untersuchung dieses Spannungsfeldes für sehr lohnend.

Unterstützung weiterer Workflow Pattern

Es ist noch zu untersuchen, welche der von van der Aalst vorgestellten Workflow Pattern noch unterstützt werden können. Bei Pattern, die nicht unterstützt werden können oder sollen muss geklärt werden, ob im klinischen Umfeld auf diese Pattern verzichtet werden kann.

Erweiterung von Designer und Engine

Flexibilität ist eines der wichtigsten Designziele. Änderung eines bestehenden Workflows werden eher die Regel, als die Ausnahme sein.

Herkömmliche Workflow Engines sind auf das vielfache Ausführen von gleichen Workflows hin optimiert. Frequente Änderung von Workflow Instanzen stellen an diese Engines also neue Anforderungen. Es wird sich zeigen, ob die bestehenden Engines den neuen Anforderungen auch gewachsen sind. In jedem Fall müssen diese Instanzen in der Workflow Engine angepasst werden.

Präsentation von Instruktionen für den Benutzer

Ein „Human task“ (BPEL4People Konstrukt) wird in ActiveBPEL über die „Inbox“ ausgewählt und bearbeitet. Wie bereits erwähnt, zeigt diese nur für die Rolle relevante Aufgaben an. Die PG hat sich die Frage gestellt wie die Instruktionen dem Benutzer sinnvoll präsentiert werden können. Standardmäßig findet eine grundlegende Gruppierung der Instruktionen in der „Inbox“ nach Ihrem Bearbeitungs-Status in Ordnern nach

- open
 - unclaimed
 - claimed
 - started
 - suspended
- closed
 - completed
 - failed
 - exited
 - obsolete
 - error

und eine zusätzliche Sortierung der Aufgaben in den einzelnen Gruppierungen nach Priorität und Ablaufzeitpunkt statt. Dies ist sinnvoll, damit der Benutzer einen Überblick über seine Aufgaben bekommt. Um jetzt aber die vielleicht sehr zahlreichen Instruktionen eines Aktionsplanes zu gruppieren, reicht das nicht aus, denn jetzt muss der Benutzer eventuell immer noch viele einzelne Instruktionen „claimen“ und bearbeiten, obwohl einige davon logisch zusammenhängen - das kann recht aufwendig werden!

Als sinnvolle Erweiterung des Inbox-Konzepts stellt sich die PG daher einen Mechanismus vor, der beim Auswählen einer Instruktion, automatisch alle anderen relevanten Instruktionen des zugehörigen Aktionsplans durch Checkboxen mitauswählt. Der Benutzer kann dann immer noch einzelne Instruktionen deselektieren. Da das den Rahmen einer Machbarkeitstudie zum PG-Thema verlässt, wurde eine solche Modifikation der „Inbox“ nicht realisiert, wäre aber für eine erfolgreiche Umsetzung sinnvoll. Schließlich ist in dem PG-Konzept angedacht, dass nur der verantwortliche Arzt eine ad hoc Änderung durchführt. Andere Benutzer müssen also nicht zwingend den Designer nutzen und sind eventuell auf eine übersichtliche Präsentation ihrer Aufgaben angewiesen. Abbildung 8.1 stellt dar, wie eine entsprechende Lösung aussehen könnte.

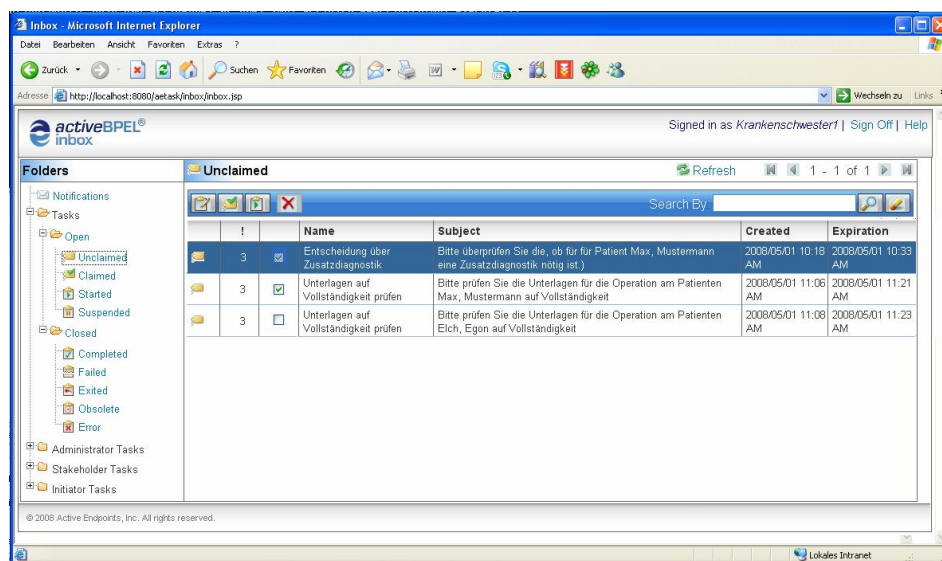


Abbildung 8.1: Gruppierte Instruktionen eines Aktionsplanes .

Diese Auswahl der Instruktionen kann dann komplett beansprucht („geclaimed“) und bearbeitet werden. Bei der Bearbeitung werden die Formulare der einzelnen Aufgaben dann aneinander gereiht und können so als Ganzes bearbeitet werden. Dabei werden zu jeder Aufgabe die entsprechenden Kopfzeilen, wie es in Abbildung 8.2 dargestellt ist, angezeigt.

8.3 Danksagung

Wir möchten uns bei unseren Betreuern des Lehrstuhls Martin und Markus, bei unseren Betreuern vom ISST (Claudia, Jan, Markus) und natürlich bei Prof. Rehof bedanken, dass sie diese Projektgruppe möglich gemacht haben. Ohne die außergewöhnlich tatkräftige Unterstützung von Sebastian und Sven-Torben wären wir in unserer Arbeit nie so weit gekommen.

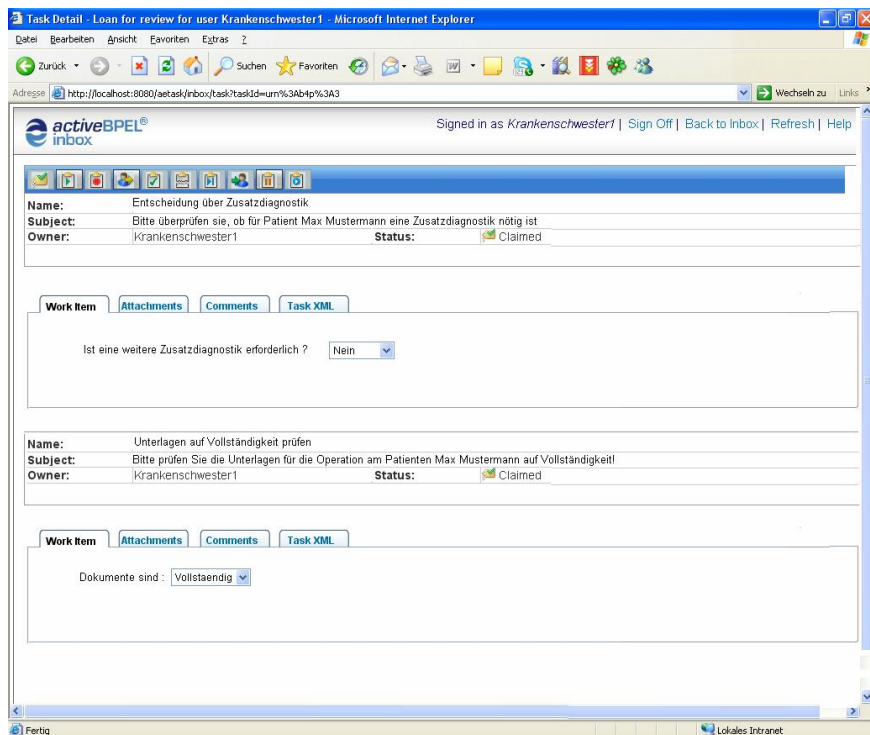


Abbildung 8.2: Formulare der gruppierten Instruktionen eines Aktionsplanes .

Literaturverzeichnis

- [1] AgilPro. <http://www.agilpro.eu/>.
- [2] BABEL Tools. <http://www.bpm.fit.qut.edu.au/projects/babel/tools/>.
- [3] DICOM Homepage. <http://medical.nema.org>.
- [4] Eclipse BPEL Project. <http://www.eclipse.org/bpel/index.php>.
- [5] Eclipse BPMN Modeler. <http://www.eclipse.org/stp/bpmn/>.
- [6] Fraunhofer SPOT index Startseite. <http://www.spot.fraunhofer.de/>.
- [7] Help - ActiveVOS. <http://www.activebpel.org/infocenter/ActiveVOS/v50/index.jsp?topic=/com.activee.bpel.doc/html/UG3.html>.
- [8] Help - ActiveVOS. <http://infocenter.activevos.com/infocenter/ActiveVOS/v60/index.jsp?topic=/com.activee.bpel.doc/html/UG9-11.html>.
- [9] Help - ActiveVOS. <http://infocenter.activevos.com/infocenter/ActiveVOS/v60/index.jsp?topic=/com.activee.bpel.doc/html/UG10-4.html>.
- [10] HL7 Benutzergruppe in Deutschland e.V. http://www.hl7.de/standard/wasist_hl7.php.
- [11] HL7 Standards. http://www.hl7.org/Library/standards_non1.htm.
- [12] IBM Pattern Solutions : Use patterns to drive productivity in software design and development. <http://www.ibm.com/developerworks/rational/products/patternsolutions/>.
- [13] Intalio BPMN Designer. <http://www.intalio.com/products/designer/>.
- [14] Intalio Server. <http://www.intalio.com/products/server/>.
- [15] Intalio Workflow. <http://www.intalio.com/products/workflow/>.
- [16] Oracle BPEL Designer. <http://www.oracle.com/technology/products/ias/bpel/index.html>.
- [17] Petri-Netz. <http://de.wikipedia.org/wiki/Petrinetz>.

-
- [18] PG DynamicFlow. <http://ls14-www.cs.uni-dortmund.de/opencms/de/lehre/projektgruppen/dynamicflow.html>.
- [19] SPOT Modeling Language- Die Elemente. http://www.spot.fraunhofer.de/fhg/Images/SPOT_Modeling-Language_tcm421-133121.pdf.
- [20] Universität Augsburg. <http://www.informatik.uni-augsburg.de/en/chairs/swt/ds/>.
- [21] Wartezeiten in der Notaufnahme drastisch verkürzt - Modell Asklepios Klinik Hamburg-Altona. <http://www.lifepr.de/pressemeldungen/deutsche-gesellschaft-interdisziplinaere-notfallaufnahme-dgina-ev/boxid-22411.html>.
- [22] Workflow Patterns - Patterns - Control - Structured Discriminator. http://www.workflowpatterns.com/patterns/control/advanced_branching/wcp9.php.
- [23] *Modellierung, Planung und Ausführung klinischer Pfade - Eine Integrierte modellbasierte Betrachtung evidenzbasierter organisatorischer und betriebswirtschaftlicher Gesichtspunkte*. ibidem-Verlag, 2005.
- [24] active endpoints. Homepage active endpoints. <http://www.activevos.com/>.
- [25] ärzteblatt.de. Internisten fordern Qualitätssicherung in der Gesundheitspolitik. <http://www.aerzteblatt.de/v4/news/news.asp?p=Leitlinien&src=suche&id=30286>. 05.03.2008 14:00.
- [26] ärzteblatt.de. Internisten richten Amt eines Leitlinienbeauftragten ein. <http://www.aerzteblatt.de/v4/news/news.asp?p=Leitlinien&src=suche&id=31445>. 05.03.2008 14:00.
- [27] AWMF. Homepage der AWMF. <http://www.awmf.org>, O.J. 04.02.2008 11:00.
- [28] AWMF. Leitlinien Homepage der AWMF. http://www.uni-duesseldorf.de/awmf/11/11_list.htm, O.J. 04.02.2008 11:00.
- [29] ÄZQ. Homepage des ÄZQ. <http://www.aezq.de/aezq/0index/view>, O.J. 04.02.2008 11:00.
- [30] J.C.M. Baeten. A Brief History of Process Algebra.
- [31] M. Greiling; M. Hessel; K. Berger, editor. *Pfadmanagment im Krankenhaus - Führen mit Kennzahlensystemen*. Verlag W. Kohlhammer, Stuttgart, 2004.
- [32] S. Brockmann. *Evidenzbasierte hausärztliche Leitlinien: Entwicklung, konzeptionelle Probleme und praktische Lösungen auf dem Hintergrund ihrer Ideengeschichte*. PhD thesis, Heinrich-Heine-Universität Düsseldorf, 2003.

- [33] C. Bunse. *Pattern-based Refinement and Translation of Object-Oriented Models to Code*. Fraunhofer-Institut Experimentelles Software Engineering, 2001.
- [34] N. Roeder; H. Bunzemeier, editor. *Kompendium zu G-DRG-System 2007*. Deutsche Krankenhaus Verlagsgesellschaft mbH, Düsseldorf, 2007.
- [35] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, 1991.
- [36] DEGAM. Homepage der DEGAM. <http://www.degam.de/>, O.J. 04.02.2008 11:00.
- [37] Dr. med. Petra Rudolph. Persönliches Gespräch. Fachärztin für Gynäkologie und Geburtshilfe.
- [38] A. Agrawal et al. WS-BPEL Extension for People (BPEL4People), Version 1.0. Technical report, June 2007.
- [39] B. Benatallah et al. HiWorD: A Petri Net-Based Hierarchical Workflow Designer. In *ACSD '03: Proceedings of the Third International Conference on Application of Concurrency to System Design*, page 235, Washington, DC, USA, 2003. IEEE Computer Society.
- [40] B. Matjaz et al., editor. *Business Process Execution Language for Web Services, Second Edition*. Packt Publishing Ltd., Birmingham, 2006.
- [41] C. Alexander et al. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, USA, 1977.
- [42] C. Schneider et al. Leitlinienadäquate Kenntnisse von Internisten und Allgemeinmedizinern am Beispiel der arteriellen Hypertonie. <http://www.elsevier.de/elsevier/journals/files/zaefq/archive/501/339.pdf>, 2001. 04.02.2008 19:00.
- [43] C.M. Boyd et al. Clinical Practice Guidelines and Quality of Care for Older Patients With Multiple Comorbid Diseases. <http://jama.ama-assn.org/cgi/content/short/294/6/716>. 05.03.2008 14:00.
- [44] D. L. Sackett et al. Was ist EbM und was nicht? <http://www.ebm-netzwerk.de/grundlagen/wasistebm>, 2006. 04.03.2008 19:00.
- [45] D. Steinberg et al. *EMF Eclipse Modelling Framework*. Addison Wesley, second edition edition, 2009.
- [46] J. Hidders et al. DFL: A dataflow language based on Petri nets and nested relational calculus. *Inf. Syst.*, 33(3):261–284, 2008.
- [47] M. Adams et al. Facilitating Flexibility and Dynamic Exception Handling in Workflows through Worklets. In O. Belo et al., editor, *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05), Porto, Portugal, 13-17 June, 2005*,

- CAiSE Forum, Short Paper Proceedings*, volume 161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
- [48] M. Kloppmann et al. WS-BPEL Extension for People - BPEL4People. A joint white paper by ibm and sap, July 2005.
- [49] M. Peleg et al. Guideline Interchange Format 3.5 Technical Specification. Technical report, InterMed Collaboratory, 2004.
- [50] N. Roeder et al. Frischer Wind mit klinischen Behandlungspfaden I. Instrumente zur Verbesserung der Organisation klinischer Prozesse - News und Trends. *Das Krankenhaus*, 01:20–27, 2003.
- [51] N. Roeder et al. Frischer Wind mit klinischen Behandlungspfaden II. Instrumente zur Verbesserung der Organisation klinischer Prozesse - News und Trends. *Das Krankenhaus*, 02:124–130, 2003.
- [52] N. Roeder et al., editor. *Klinische Behandlungspfade in der inneren Medizin - Am Beispiel der akut-stationären Rheumatologie*. Deutscher Ärzte-Verlag GmbH, Köln, 2007.
- [53] N. Russell et al. Workflow Data Patterns. QUT Technical report FIT-TR-2004-01, Queensland University of Technology, 2004.
- [54] N. Russell et al. Workflow Resource Patterns. BETA Working Paper Series 127, Eindhoven University of Technology, 2004.
- [55] N. Russell et al. Workflow Control-Flow Patterns : A Revised View. BPM Center Report BPM-06-22, BPMcenter.org, 2006.
- [56] P. Wohed et al. Pattern-based Analysis of UML Activity Diagrams. BETA Working Paper Series 129, Eindhoven University of Technology, 2004.
- [57] R.H. Pantell et al. Management and Outcomes of Care of Fever in Early Infancy . <http://jama.ama-assn.org/cgi/reprint/291/10/1203.pdf>. 05.03.2008 14:00.
- [58] S.W. Sadiq et al. Data flow and validation in workflow modelling. In *ADC '04: Proceedings of the 15th Australasian database conference*, pages 207–214, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
- [59] S.W. Sadiq et al. Specification and validation of process constraints for flexible workflows. *Inf. Syst*, 30(5):349–378, 2005.
- [60] E. Nagel; C. Fuchs, editor. *Leitlinien und Standards im Gesundheitswesen - Fortschritt in sozialer Verantwortung oder Ende der Ärztlichen Therapiefreiheit*. Deutscher Ärzte-Verlag GmbH, Köln, 1997. Dokumentation des Wissenschaftlichen Symposiums 17-18. März 1995 in Mainz.

- [61] Ärztliches Zentrum für Qualität in der Medizin. Das deutsche Leitlinien-Clearingverfahren 1999-2005. -Hintergrund, Zielsetzung, Ergebnisse - Abschlussbericht. http://www.q-m-a.de/azq/azq/content/publikationen/0index/schriftenreihe/pdf/abschluss_cv.pdf, 2006. 04.02.2008 19:00.
- [62] Ärztliches Zentrum für Qualität in der Medizin. Handbuch zur Entwicklung regionaler Leitlinien. <http://www.q-m-a.de/azq/azq/content/publikationen/0index/schriftenreihe/pdf/schriftenreihe26.pdf>, 2006. 04.02.2008 20:00.
- [63] M. Greiling, editor. *Pfade durch das klinische Prozessmanagement - Methodik und Aktuelle Diskussion*. Verlag W. Kohlhammer, Stuttgart, 2004.
- [64] D. Hart, editor. *Ärztliche Leitlinien im Medizin und Gesundheitsrecht - Recht und Emirie professioneller Normbildung*. Nomos, Baden-Baden, 2005.
- [65] M. Havey, editor. *Essential Business Process Modeling*. O'Reilly, Sebastopol, 08/2005.
- [66] S. Meinecke Fraunhofer ISST (Hrsg). *Umsetzung des Klinischen Pfades: Ambulante Knie-OP*. Fraunhofer ISST, 2008.
- [67] SAP IBM. Bpel4People Specification 1.0. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf.
- [68] SAP IBM. Bpel4People Whitepaper. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_white_paper.pdf.
- [69] Intalio. Homepage Intalio. <http://www.intalio.com/>.
- [70] Intalio. Homepage Intalio Tempo. <http://www.intalio.org/confluence/display/TEMPO/>.
- [71] Ihle J. *Ärztliche Leitlinien, Standards und Sozialrecht*. Nomos, Baden-Baden, 2006.
- [72] M. Peleg; R. Kantor. Approaches for Guideline Versioning Using GLIF. Technical report, Stanford Medical Informatics, Division of Infectious Diseases and Center for AIDS Research, Stanford University School of Medicine, Stanford, CA, O.J.
- [73] N. Roeder; T. Küttner, editor. *Klinische Behandlungspfade - Mit Standards erfolgreich Arbeiten*. Deutscher Ärzte-Verlag GmbH, Köln, 2007.
- [74] J. Meyer. Anforderungen an zukünftige Workflow-Management-Systeme: Flexibilisierung, Ausnahmebehandlung und Dynamisierung - Erörterung am Beispiel medizinischorganisatorischer Abläufe. Master's thesis, Universität Ulm, 1996.
- [75] K.H. Vosteen; W. Müller, editor. *Vorläufige Übersicht der elektronisch publizierten Leitlinien für Diagnostik und Therapie der Wissenschaftlichen Medizinischen Fachgesellschaften - Stand 23. Oktober 1998.*, volume 1. AWMF, Düsseldorf, 1998.

- [76] K.H. Vosteen; W. Müller, editor. *Vorläufige Übersicht der elektronisch publizierten Leitlinien für Diagnostik und Therapie der Wissenschaftlichen Medizinischen Fachgesellschaften - Stand 23. Oktober 1998.*, volume 2. AWMF, Düsseldorf, 1998.
- [77] K.H. Vosteen; W. Müller, editor. *Vorläufige Übersicht der elektronisch publizierten Leitlinien für Diagnostik und Therapie der Wissenschaftlichen Medizinischen Fachgesellschaften - Stand 23. Oktober 1998.*, volume 3. AWMF, Düsseldorf, 1998.
- [78] K.H. Vosteen; W. Müller, editor. *Vorläufige Übersicht der elektronisch publizierten Leitlinien für Diagnostik und Therapie der Wissenschaftlichen Medizinischen Fachgesellschaften - Stand 23. Oktober 1998.*, volume 4. AWMF, Düsseldorf, 1998.
- [79] J. Nitzsche. Entwicklung eines Monitoring-Tools zur Unterstützung von parametrisierten Web Services Flow. Master's thesis, Universität Stuttgart, 2006.
- [80] Apache ODE. Homepage Apache ODE. <http://ode.apache.org/>.
- [81] Apache Org. Apache Tomcat Admin Pack. <http://archive.apache.org/dist/tomcat/tomcat-5/v5.5.12/bin/>.
- [82] Apache Org. Homepage Apache Axis2 Projekt. <http://ws.apache.org/axis2/>.
- [83] Parametrix Solutions. *Phoenix - Parametrierleitfaden*. MCS Parametrix. Version 2(5.9.11) 02/2005.
- [84] C.A. Petri. Kommunikation mit Automaten. *Schriften des ReinschWestfalischen Inst. für Instrumentelle Mathematik an der Un.Bonn, Bonn FRG*, 1962.
- [85] R. Müller; R. Greiner; E. Rahm. AgentWork: A Workflow System Supporting Event-Oriented Workflow Adaption. Technical report, Department of Computer Science, University of Leipzig, Germany, O.J.
- [86] M. Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. PhD thesis.
- [87] P. Dadam; M. Reichert. Adeptflex : supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 1998.
- [88] P. Dadam; M. Reichert. Clinical Workflows – The Killer Application for Process-oriented Information Systems? *Abramowicz W, Orlowska ME (Eds.): BIS 2000 – Proc. of the 4th Int'l Conference on Business Information Systems in Poznan, Poland, April 2000*, 2000.
- [89] C. Reuter. Realisierung klinischer Pfade. Master's thesis, Universität Erlangen-Nürnberg, Department Informatik, Lehrstuhl für Informatik 6 (Datenmanagement), 2004.
- [90] J. Robertson, S.; Robertson. *Mastering the requirements process*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

- [91] H. Sandholzer, editor. *Praxistrainer Allgemeinmedizin - Leitlinienbasierte Fallseminare*. Schattauer, Stuttgart, 2007.
- [92] T. Boecker; A. Smolarczyk. Seminarthema - ADEPT : Dynamische Ablaufänderungen mit KF-Graphen). Erstellt im Rahmen der PG DynamicFlow.
- [93] Intalio Open Source. Intalio Open Source. <http://www.intalio.com/company/open-source/>.
- [94] AJDT Team. Homepage ajdt projekt. <http://www.eclipse.org/ajdt/>.
- [95] unknown. Definition EbM. <http://www.ebm-netzwerk.de/grundlagen/definitionen>, 2006. 04.03.2008 19:00.
- [96] unknown. Geschichte der EbM. <http://www.ebm-netzwerk.de/grundlagen/geschichte>, 2006. 04.03.2008 19:00.
- [97] M. Voorhoeve; W.M.P. van der Aalst. Ad-hoc workflow: problems and solutions. *Database and Expert Systems Applications, 1997. Proceedings., Eighth International Workshop on*, 1(2):36 – 40, September 1997.
- [98] N. Russell; W.M.P. van der Aalst; A.H.M. ter Hofstede. Exception Handling Patterns in Process-Aware Information Systems. BPM Center Report BPM-06-04, BPMcenter.org, 2006.
- [99] W.M.P. van der Aalst; A.H.M. ter Hofstede. YAWL: Yet another workflow language.
- [100] W.M.P. van der Aalst; A.H.M. ter Hofstede. YAWL: Yet another workflow language (revised version). QUT Technical report, FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.
- [101] J. Vanhatalo. BPEL Repository User Guide. <http://dl.alphaworks.ibm.com/technologies/bpelrepository/BPEL-Repository-User-Guide-2.1.0.pdf>, 2006. 2008.
- [102] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Verlag, first edition, November 2007.
- [103] D. Bradshaw; M. Kennedy; C. West. Oracle BPEL Process Manager Developer's Guide 10g Release 2 (10.1.2). Developer's Guide 2, Oracle BPEL Process Manager development, product management, and quality assurance teams, October 2005.
- [104] L. Wittgenstein. *Logisch-philosophische Abhandlung, Tractatus logico-philosophicus*. Kegan Paul, Trench, Trubner & Co., 1922.
- [105] D. Wutke. Erweiterung einer Workflow-Engine zur Unterstützung von parametrisierten Web Service Flows. Master's thesis, Institut für Architektur von Anwendungssystemen Stuttgart, 2006.

A Beschreibung des Beispielworkflows

A.1 Beschreibung der Aktionspläne

Hier werden die einzelnen Aktionspläne beschrieben, indem die jeweils Ablaufbedingungen, Instruktionen und Überwachungsregeln aufgelistet werden. Die Ablaufbedingungen werden für jeden Aktionsplan tabellarisch dargestellt. Die Tabelle enthält drei Spalten mit je einer Regel (SKIP-, ABORT-, und WAIT-Regel). Die einzelnen Variablen, die in den Regeln benutzt werden sind in Anhang A.3 genauer beschrieben. Nach den Ablaufbedingungen werden die Instruktionen des Aktionsplans aufgelistet. Sie werden mit der folgenden Form dargestellt:

APX-Y: Name_Instruktion(var₁, var₂, ...)
Vorher beendet: APX₁-Y₁, APX₂-Y₂, ...

- *APX*: steht für einen Aktionsplan, z.B. AP01 für die Anamnese.
- *Y*: steht für die Nummer der Instruktion innerhalb des Aktionsplans.
- *Name_Instruktion*: Name der Instruktion mit der Nummer *Y*.
- (*var₁, var₂, ...*): Liste der benötigten Variablen (siehe Anhang A.3), die bei der Instruktion *Name_Instruktion* eingegeben werden müssen (die Reihenfolge der Eingaben ist willkürlich).
- *Vorher beendet: APX₁-Y₁, APX₂-Y₂, ...*: Liste der innerhalb des Aktionsplans betrachteten Instruktionen, die vor Ausführung der Instruktion mit der Nummer *Y* schon vorher beendet werden müssen. Die Reihenfolge ergibt sich hier trivialerweise aus der Beschreibung (*Vorher beendet*-Liste) der einzelnen Instruktionen nämlich von APX₁-Y₁, APX₂-Y₂, ...
- *-/-*: steht für keine Angabe.

A.1.1 AP01: Anamnese

Ablaufbedingungen

SKIP	ABORT	WAIT
-/-	-/-	$pav \wedge usv$

Instruktionen

AP01-01: Patient aufrufen(*daa, pat, wza, ura*)
Vorher beendet: -/-

AP01-02: Patientenakten bereitstellen(*daa, medium, patDat*)
Vorher beendet: AP01-01

AP01-03: Digitalen Antwortbogen bearbeiten(*daa, formAnamnese*)
Vorher beendet: AP01-01, AP01-02

AP01-04: Digitalen Antwortbogen bearbeiten(*daa, formSozAnamnese*)
Vorher beendet: AP01-01, AP01-02

AP01-05: Indikation berechnen(*system*)
Vorher beendet: AP01-03, AP01-04

Überwachungsregeln

- -/-

A.1.2 AP02: Telefonische Rücksprache

Ablaufbedingungen

SKIP	ABORT	WAIT
$\neg ce \wedge (alt > 3 \text{ Monate})$	-/-	<i>indKnieOpSYS</i>

Instruktionen

AP02-01: Rücksprache planen(*daa, ana*)
Vorher beendet: -/-

AP02-02: Gesprächsnotiz erstellen(*daa, ana*)
Vorher beendet: -/-

Überwachungsregeln

- -/-

A.1.3 AP03: Indikationsbewertung

Ablaufbedingungen

SKIP	ABORT	WAIT
-/-	-/-	$indKnieOpSYS \wedge pav \wedge abv \wedge sabv$

Instruktionen

AP03-01: Dokument bereitstellen(*daa, medium, patDat*)

Vorher beendet: -/-

AP03-02: Dokument bereitstellen(*daa, medium, formAnamnese*)

Vorher beendet: -/-

AP03-03: Dokument bereitstellen(*daa, medium, formSozAnamnese*)

Vorher beendet: -/-

AP03-04: Bewerten(*daa*)

Vorher beendet: AP03-01, AP03-02, AP03-03

AP03-05: Abweichungsdokumentation erstellen(*daa, indKnieOpDA, indKnieOpSYS*)

Vorher beendet: AP03-04

Überwachungsregeln

- -/-

A.1.4 AP04: Vorbereitung der Röntgenuntersuchung

Ablaufbedingungen

SKIP	ABORT	WAIT
$roentVorh \vee befVorh$	$\neg indKnieOpDA$	-/-

Instruktionen

AP04-01: Untersuchungsart auswählen(*daa, artSet*)

Vorher beendet: -/-

AP04-02: Befundtyp Festlegen(*daa*)

Vorher beendet: AP04-01

AP04-03: Dokument drucken(*system, patDat, art, kurz, normal, druckU*)
Vorher beendet: AP04-02

AP04-04: Dokument unterschreiben(*daa, roentSch*)
Vorher beendet: AP04-03

Überwachungsregeln

- -/-

A.1.5 AP05: Anamnesebogen

Ablaufbedingungen

SKIP	ABORT	WAIT
-/-	-/-	-/-

Instruktionen

AP05-01: Dokument drucken(*system, formAnamnese, formSozAnamnese, druckU*)
Vorher beendet: -/-

AP05-02: Dokument unterschreiben(*pat, anamBog*)
Vorher beendet: AP05-01

AP05-03: Dokument in Mappe ablegen(*daa, anamBogU, prOpMap*)
Vorher beendet: AP05-02

Überwachungsregeln

- -/-

A.1.6 AP06: Radiologische Untersuchung Postphase

Ablaufbedingungen

SKIP	ABORT	WAIT
$\neg roentSchU$	-/-	$roentVorh \vee befVorh$

Instruktionen

AP06-01: Anmelden an Ambulanz(*pat*)
 Vorher beendet: -/-

AP06-02: Patient aufrufen(*daa, pat, wza, ura*)
 Vorher beendet: AP06-01

Überwachungsregeln

- -/-

A.1.7 AP07: Letzte Indikationsstellung

Ablaufbedingungen

SKIP	ABORT	WAIT
-/-	-/-	-/-

Instruktionen

AP07-01: Dokument bereitstellen(*daa, medium, patDat*)
 Vorher beendet: -/-

AP07-02: Dokument bereitstellen(*daa, medium, formAnamnese*)
 Vorher beendet: -/-

AP07-03: Dokument bereitstellen(*daa, medium, formSozAnamnese*)
 Vorher beendet: -/-

AP07-04: Dokument bereitstellen(*daa, medium, bef d*)
 Vorher beendet: -/-

AP07-05: Kurzbefund aufrufen(*system, kurz, kbef d*)
 Vorher beendet: AP07-04

AP07-06: Bewerten(*daa*)
 Vorher beendet: AP07-01, AP07-02, AP07-03, AP07-04, AP07-05

AP07-07: Entscheidung dokumentieren(*daa, indKnieOpDA*)
 Vorher beendet: AP07-06

Überwachungsregeln

- -/-

A.2 Beschreibung der Instruktionen

Hier werden die Instruktionen genauer betrachtet, wobei es für jede Instruktion und in jede Spalte den Namen der Instruktion, die davon konsumierten und die daraus erzeugten Daten (Liste der Variablen in Anhang [A.3](#)) aufgelistet werden. Unter der Tabelle befindet sich eine in Textform Erklärung, was genau in jede Instruktion passiert. Die Instruktionen sind nochmal durch die Notation *APX-Y* eindeutig gekennzeichnet.

A.2 Beschreibung der Instruktionen

Name	konsumierte Daten	erzeugte Daten
AP01-01. Patient aufrufen()	daa, pat, wza, ura	-/-
AP01-02. Dokument bereitstellen()	daa, medium, patDat	-/-
AP01-03. Digitalen Antwortbogen bearbeiten()	daa, formAnamnese	-/-
AP01-04. Digitalen Antwortbogen bearbeiten()	daa, formSozAnamnese	-/-
AP01-05. Indikation berechnen()	system	indKnieOpSYS
AP02-01. Rücksprache planen()	daa, ana	-/-
AP02-02. Gesprächsnotiz erstellen()	daa, ana	-/-
AP03-01. Dokument bereitstellen()	daa, medium, patDat	-/-
AP03-02. Dokument bereitstellen()	daa, medium, formAnamnese	-/-
AP03-03. Dokument bereitstellen()	daa, medium, formSozAnamnese	-/-
AP03-04. Bewerten()	daa	indKnieOpDA
AP03-05. Abweichungsdokumentation erstellen()	daa, indKnieOpDA, indKnieOpSYS	abwDok
AP04-01. Untersuchungsart auswählen()	daa, artSet	art
AP04-02. Befundtyp festlegen()	daa	kurz, normal
AP04-03. Dokument drucken()	patDat, art, kurz, normal, druckU	roentSch
AP04-04. Dokument unterschreiben()	daa, roentSch	roentSchU
AP05-01. Dokument drucken()	system, formAnamnese, formSozAnamnese, druckU	anamBog
AP05-02. Dokument unterschreiben()	pat, anamBog	anamBogU
AP05-03. Dokument in Mappe ablegen()	daa, anamBogU, prOpMap	-/-
AP06-01. Anmelden an Ambulanz()	pat	-/-
AP06-02. Patient aufrufen()	daa, pat, wza, ura	-/-
AP07-01. Dokument bereitstellen()	daa, medium, patDat	-/-
AP07-02. Dokument bereitstellen()	daa, medium, formAnamnese	-/-
AP07-03. Dokument bereitstellen()	daa, medium, formSozAnamnese	-/-

Name	konsumierte Daten	erzeugte Daten
AP07-04. Dokument bereitstellen()	daa, medium, befd	-/-
AP07-05. Kurzbefund aufrufen()	system, kurz, kbefd	-/-
AP07-06. Bewerten()	daa	indKnieOpDA
AP07-07. Entscheidung dokumentieren()	daa, indKnieOpDA	entDok

Erklärung der einzelnen Instruktionen:

- AP01-01: Patient wird von einem bestimmten Arzt aufgerufen und von einem Wartezimmer der Ambulanz in einem zuvor festgelegten Untersuchungsraum eingeladen, wo alle Untersuchungen des Tages durchgeführt werden.
- AP01-02: Der Arzt ruft den Patientendatensatz auf seinem Medium(z.B. Tablet-PC) auf. Patientenakte müssen vorher vorhanden sein.
- AP01-03: Zeigt dem Arzt ein xml-formular an und speichert es nach Bearbeitung. Der Arzt stellt hier Fragen zu Beschwerden, Vorerkrankungen und medikamentöser Behandlung.
- AP01-04: Zeigt dem Arzt ein xml-formular an und speichert es nach Bearbeitung. Hier werden soziale Aspekte vom Arzt erfragt.
- AP01-05: Damit berechnet das System anhand mehrerer Kriterien eine Indikation und schlägt sie dem Arzt vor. Z.B. „Im Falle einer vorliegenden chronischen Erkrankung oder einem Alter des Patienten jünger als drei Monate, schlägt das System vor, eine anästhesiologische Konsultation vorzunehmen“. Im Falle einer positiven Indikation nimmt die Variable *indKnieOpSYS* den Wert **true**.
- AP02-01: Der diensthabende Arzt der Ambulanz plant eine Rücksprache mit einem bestimmten Anästhesist und ruft ihn ggf. an.
- AP02-02: Der Arzt erstellt einen Notiz zum Gespräch zwischen ihm und dem Anästhesist.
- AP03-01: Der Arzt ruft den Patientendatensatz auf seinem Medium auf.
- AP03-02: Der Arzt ruft den Fragebogen zur Anamnese auf seinem Medium auf.
- AP03-03: Der Arzt ruft den Fragebogen zur Sozialanamnese auf seinem Medium auf.
- AP03-04: Der Arzt bewertet, ob eine ambulante Durchführung der OP. anhand der bereitgestellten Dokumente notwendig ist. Das Ergebnis der Indikation wird in der Variable *indKnieOpDA* gespeichert.
- AP03-05: Der Arzt erstellt eine Abweichungsdokumentation, indem hier die zwei Variablen *indKnieOpDA* und *indKnieOpSYS* verglichen werden. Die Abweichungsdokumentation wird z.B. in einem xml-Dokument gespeichert.
- AP04-01: Hier wird damit ausgewählt, welche Art aus einer Menge von Untersuchungsarten (z.B. Röntgen, MRT,...) durchzuführen ist. Die erzeugte Variable *art* bestimmt die ausgewählte Art.
- AP04-02: Der Arzt legt den Befundtyp fest, entweder reicht es einen normalen Befund zu erstellen oder es muss noch dazu einen kurzen festgelegt werden.

- AP04-03: Anhand Daten aus dem Patientendatensatz, der Untersuchungsart und dem Befundtyp wird ein Röntgenschein automatisch erstellt und mit dem Drucker des Untersuchungsraums ausgedruckt.
- AP04-04: Wenn der Arzt den Röntgenschein unterschreibt dann nimmt die Variable *roentSchU* den Wert *true* an.
- AP05-01: Anhand Daten aus dem Fragebogen zur Anamnese und Fragebogen zur Sozialanamnese wird ein Anamnesebogen automatisch erstellt und mit dem Drucker des Untersuchungsraums ausgedruckt.
- AP05-02: Wenn der Patient den Anamnesebogen unterschreibt dann nimmt die Variable *anamBogU* den Wert *true* an.
- AP05-03: Der untergeschriebene Ausdruck des Anamnesebogens wird vom Arzt in die präoperative Mappe gelegt.
- AP06-01: Der Patient meldet sich nach der radiologischen Untersuchung wieder an Ambulanz an.
- AP06-02: siehe AP01-01.
- AP07-01: siehe AP03-01.
- AP07-02: siehe AP03-02.
- AP07-03: siehe AP03-03.
- AP07-04: Der Arzt ruft den Befund auf sein Medium auf.
- AP07-05: Sobald der Arzt den Befund aufruft, prüft das System durch die Variable *kurz*, ob es einen Kurzbefund gibt. Wenn ja, dann ruft das System automatisch den Kurzbefund auf dem Medium vom Arzt auf.
- AP07-06: Der Arzt bewertet nochmal die Indikationsstellung anhand der bereitgestellten Dokumente. Das Ergebnis der Indikation überschreibt die Variable *indKnieOpDA*.
- AP07-07: Falls die Indikation negativ ist, erstellt der Arzt ein Entscheidungsdocument, indem er seine Entscheidung anhand geeigneter Multiple-Choice-Antworten begründet.

A.3 Liste der Variablen

Hier werden alle vorher benutzten (konsumierten und erzeugten) Daten aufgelistet. Alle Abkürzungen sind genauer beschrieben. Außerdem ist jede Variable mit ihrem Typ gekennzeichnet.

abv := Anamnesebogen vorhanden <boolean>
abwDok := Abweichungsdokument <xml-Doc>
alter := Alter des Patienten (z.B. laut elektr. Patientenakte) <int>
ana := Anästhesist <string>
anamBog := Anamnesebogen <txt-Doc>
anamBogU := Anamnesebogen vom Patient untergeschrieben <boolean>
art := Untersuchungsart <String>
artSet := Menge aller möglichen Untersuchungsarten <string[]>
befd := Befund der Röntgenuntersuchung <xml-Doc>
befVorh := Befund des Patients ist vorhanden <boolean>
ce := Chronische Erkrankung vorhanden <boolean>
daa := Diensthabender Arzt der Ambulanz <string>
druckU := Drucker des Untesuchungsraums <printer>
entDok := Entscheidungsdokument im Falle einer negativen Indikation <txt-Doc>
formAnamnese := Fragebogen zur Anamnese <xml-FormularVorlage>
formSozAnamnese := Fragebogen zur Sozialanamnese <xml-FormularVorlage>
indKnieOpDA := Indikationsbewertung ambulante Knie-Op - Arzt <boolean>
indKnieOpSYS := Indikationsbewertung ambulante Knie-Op - angeschl. Experten-System
<boolean>
kbefd := kurzer Befund nach der radiologischen Untersuchung <xml-Doc>
kurz := kurzer Befund ist zu ertellen <boolean>
medium := Medium um Text anzuzeigen <string: „Drucker“, „Bildschirm“, „PDA“, ...>
normal := normale Bfandung ist zu erstellen <boolean>
pat := zu behandelter Patient <string>
patDat := Patientendatensatz <xml-Doc>
pav := Patientenakte vorhanden <boolean>
prOpMap := präoperative Mappe <mappe>

roentSch := automatisch erstellter und ausgedruckter Röntgenschein <txt-Doc>

roentSchU := Röntgenschein ist vom Arzt untergeschrieben <boolean>

roentVorh := Röntgenbilder des Patients sind vorhanden <boolean>

sabv := Sozialanamnesebogen vorhanden <boolean>

system := Experten-System <string>

ura := Untersuchungsraum der Ambulanz <string>

usv := Überweisungsschein eines überweisenden Arztes vorhanden <boolean>

wfNow := Aktuelle Uhrzeit <double>

wfSStart := Startzeitpunkt des Workflows <double>

wza := Wartezimmer der Ambulanz <string>