
DISSERTATION

Plattform zur integrierten IT-gestützten Ausführung von strikten und agilen Prozessen

Michael Zeising

Von der Universität Bayreuth zur Erlangung des
Grades eines *Doktors der Naturwissenschaften* (Dr.
rer. nat.) genehmigte Abhandlung

<i>Vorgelegt von:</i>	Michael Zeising, M.Sc. aus Leipzig
<i>Erster Gutachter:</i>	Prof. Dr.-Ing. Stefan Jablonski
<i>Zweiter Gutachter:</i>	Prof. Dr. Torsten Eymann
<i>Tag der Einreichung:</i>	27. Mai 2015
<i>Tag des Kolloquiums:</i>	21. Oktober 2015

Für Maria.

Zusammenfassung

Die Methodik des Prozessmanagements ist aus unterschiedlichsten Richtungen motiviert und hilft verschiedensten Organisationen sich intern zu strukturieren. Die Unterstützung von Prozessen durch die maschinelle Interpretation von Prozessmodellen bildet das ehrgeizigste Ziel des Prozessmanagements und gewinnt stetig an Bedeutung. Solche Prozessausführungssysteme (PAS) können auf der Grundlage von Modellen an Aufgaben erinnern, sie den Beteiligten zuweisen, Informationen im richtigen Moment erfragen und bereitstellen, mit integrierten Systemen kommunizieren und relevante Kennzahlen und Messgrößen erfassen. Die Modelle der Prozesse einer Organisation bilden die Konfiguration eines PAS. Entwickelt sich die Organisation weiter, so muss zur Anpassung ihrer IT-Unterstützung im Idealfall nicht die Software selbst sondern lediglich die entsprechenden Prozessmodelle angepasst werden.

Bisherige Ansätze beim Entwurf von PAS zielen vor allem auf die Unterstützung sogenannter strikter Prozesse ab, also auf solche, deren genauer Ablauf sowohl im Vordergrund steht als auch vollständig festgelegt werden kann. Sogenannte agile Prozesse hingegen zeichnen sich dadurch aus, dass ihr zeitlicher Ablauf nicht im Vordergrund steht. Vielmehr soll eine hohe Zahl an alternativen Pfaden entlang gewisser Randbedingungen ermöglicht werden. Die IT-Unterstützung solch agiler Prozesse ist bislang nur unzureichend gelöst. Da die Prozesslandschaft einer Organisation jedoch stets beide Arten von Prozessen aufweisen wird, können aktuelle Ansätze der prozessbasierten IT-Unterstützung eine Organisation demnach nur teilweise erfassen. Um das gesamte derzeit bekannte Spektrum von Prozessen durch ein IT-System unterstützen zu können, ist somit ein PAS notwendig, das sowohl strikte als auch agile Prozesse maschinell interpretiert.

Im Rahmen der vorliegenden Arbeit wird ein solches Konzept zur Unterstützung sowohl von strikten als auch von agilen Prozessen entwickelt. Dazu werden eine neuartige Form der Modellierung agiler Prozesse und das entsprechende Software-Modul zur Interpretation entworfen. Zur Modellierung und Interpretation strikter Prozesse wird auf bewährte Konzepte zurückgegriffen. Beide Module werden auf innovative Art kombiniert und ergeben ein neuartiges PAS, das zur Unterstützung des gesamten Prozessspektrums einer Organisation dienen kann. Um schließlich die Zweckdienlichkeit des Ansatzes zu zeigen, wird er anhand eines etablierten Rahmenwerks zur Beurteilung prozessbasierter Systeme evaluiert.

Abstract

The method of (Business) Process Management is motivated from various directions and helps all different organizations to structure themselves. The support of processes through the interpretation of process models forms the communities' ambitious aim and is continually increasing in importance. Based on process models, such Process-Aware Information Systems (PAIS) may remind users of tasks, assign them to participants, request information at the right moment and present them again, communicate with other systems and gather relevant key figures and indicators. An organization's processes form the configuration of a PAIS. As the organization evolves, ideally, only the process models have to be adjusted in order to realign the IT support.

Existing approaches of PAIS primarily aim at supporting so-called strict processes, i.e., those whose exact procedure is of main interest and can be specified completely. So-called agile processes, however, are characterized by the fact that their exact procedure is not a prime concern. On the contrary, a large number of alternative paths along certain boundary conditions shall be facilitated. The IT support for such agile process has been solved only unsatisfactorily. As an organization's process landscape always comprises both kinds of processes, existing approaches of process-based IT support may only partially cover an organization. In order to support the full currently known spectrum of processes, a PAIS must be capable of interpreting both strict and agile processes.

During this work, a concept for supporting strict as well as agile processes is developed. For this purpose, a novel method of modelling agile processes together with a software module for its interpretation is designed. For modelling and interpreting strict processes, established concepts are applied. Both modules are integrated in an innovative manner and result in a new type of PAIS which suffices for supporting the full spectrum of an organization's processes. In conclusion, in order to validate the usefulness of the approach, it is evaluated on the basis of an established framework for the assessment of PAIS.

Inhalt

1	Problemstellung und Zielsetzung	1
1.1	Methodik des Prozessmanagements.....	1
1.2	Prozessmanagement als Weiterentwicklung von Softwaresystemen	4
1.3	Strikte und agile Prozesse.....	5
1.4	Methodik und Zielsetzung.....	14
1.5	Lösungsskizze und Beitrag.....	15
2	Verwandte Arbeiten zur IT-gestützten Ausführung von Prozessmodellen	23
2.1	Prozedurale Modellierung.....	23
2.2	Regelbasierte Modellierung	31
2.3	Hybride Modellierungskonzepte	34
2.4	Zusammenfassung.....	36
3	Gemeinsame Prozessinfrastruktur (CPI) der Process Navigation Plattform	38
3.1	Vorbemerkung zu objektorientierten Entwurfsmustern	39
3.2	Grundsätzliches Ausführungsprinzip	39
3.3	Bewertungen und Regeln	43
3.4	Dateiverwaltung	44
3.5	Organisationsverwaltung.....	46
3.6	Aufgabenverwaltung	49
3.7	Bereitstellung von Prozessinstanzen und Aufzeichnung von Protokollen	50
3.8	Webbasierte Teilnehmeroberfläche.....	51
3.9	Zusammenfassung und Bewertung	53
4	Prozedurale Prozess-VM (PPVM).....	54
4.1	Ausdruckssprache	56
4.2	Integration von BPMN	57
4.3	Zusammenfassung.....	61
5	Regelbasierte Prozess-VM (RPVM) – Ereignisbasierte Ausführung von Prozessen.....	62
5.1	Auslöser.....	63
5.2	Erzeugung von Ereignissen.....	64
5.3	Bewertung von Ereignissen und Instanzen.....	65
5.4	Beispiel	67

5.5	Zwei Facetten einer Prozessregel.....	69
5.6	Auswertung von Regeln.....	71
5.7	Ereignisbehandlung.....	75
5.8	Mehrfach instanziiierbare Elemente	76
5.9	Resultierende Architektur der RPVM	76
5.10	Zusammenfassung.....	78
6	Deklarative Prozesszwischensprache (DPIL)	80
6.1	Abstrakte Syntax der Modellelemente	80
6.2	Ereignisse und Lebenszyklen der Modellelemente	82
6.3	Spezialisierte abstrakte Syntax für Regeln	85
6.4	Integrierte konkrete Syntax zur regelbasierten Prozessmodellierung	88
6.5	Generierung von DRL aus dem DPIL-Modell.....	94
6.6	Sprachinfrastruktur und integrierte Entwicklungsumgebung für DPIL-Prozesse	95
6.7	Zusammenfassung.....	97
7	Realisierung von Prozessmustern in DPIL und Abbildung von anderen Sprachen	98
7.1	Realisierung kontrollflussbezogener Muster	98
7.2	Realisierung datenbezogener Muster	103
7.3	Realisierung organisatorischer Muster	107
7.4	Gegenseitige Ergänzung prozedurale und regelbasierter Modellierung	111
7.5	Funktionale und operationale Muster.....	112
7.6	Bibliothek von Modellierungsmustern	112
7.7	DPIL und DECLARE	113
7.8	Abbildung von CMMN auf DPIL	116
7.9	Zusammenfassung und Bewertung	120
8	Zusammenfassung und zukünftige Aufgabenfelder	121
8.1	Migration von Prozessinstanzen auf geänderte Prozessmodelle	121
8.2	Modellierung und Ausführung von mehrfach instanziiierbaren Aktivitäten.....	122
8.3	Grafische Darstellung regelbasierter Prozessmodelle	122
8.4	Prozedurale und regelbasierte Modellierung in einer Sprache	123
8.5	Vorausschau und Simulation von Ereignisketten	124

Abbildungen

Konsolidierter Prozesslebenszyklus nach (Meerkamm 2011)	2
Exemplarisches Prozessmodell als Flussdiagramm	3
Historische Entwicklung von PAS (vgl. (van der Aalst 1996)).....	5
Exemplarischer Prozess in prozeduraler und regelbasierter Notation	9
Prozessmodell in der BPMN-Notation mit den fünf grundlegenden Perspektiven	11
Historische Entwicklung wichtiger Standards im Prozessmanagement nach (Bartonitz 2009)	13
Vereinfachter Aufnahmeprozess einer psychiatrischen Station aus strikten und agilen Anteilen	14
Ausführung prozeduraler Prozessmodelle	17
Ausführungsprinzip der <i>Rule-based Process Virtual Machine</i> (RPVM)	19
Angestrebte Gesamtarchitektur.....	21
Exemplarisches Prozessmodell als Flussdiagramm	24
Exemplarisches EPK-Modell	25
Exemplarisches BPMN-Diagramm.....	26
Einfaches BPMN-Modell und seine Entsprechung als Petri-Netz	29
Grafische Darstellung der response-Regel in DECLARE	32
Automat zum DECLARE-Beispiel.....	32
Integration von Geschäftsregeln in einen BPMN-Prozess (vgl. (Stiehl 2013))	36
Aufbau der PN-Plattform mit CPI	38
Kern des Process Virtual Machine-Musters nach Baeyens und Faura (vgl. (Baeyens & Faura 2007))	40
Ausführungszeiger der CPI für zusammengesetzte Prozessmodelle.....	40
Direkte Ausführung von BPMN-Modellen durch Generierung von Klassen aus dem XML-Schema und Deserialisierung von Modellobjekten.	41
Ablauf der Auflösung eines verschachtelten Prozessmodells	43
Schnittstelle der CPI zu Regeln und Bewertungen	44
Kernelemente des CMIS-Datenmodells (vgl. (OASIS 2012))	45
Schnittstelle der CPI zur Organisationsverwaltung und mögliche LDAP-Implementierung.....	47
Exemplarischer Verzeichnisbaum eines LDAP-Verzeichnisses	48
Entsprechung des exemplarischen Verzeichnisbaums im Rahmen der CPI (die Assoziationen „subject“, „predicate“ und „object“ sind mit „s“, „p“ und „o“ abgekürzt).....	48
Schnittstelle der Aufgabenverwaltung	49
Schnittstelle zur Verwaltung von Instanzen und zum Logging	50
Webbasierte Teilnehmeroberfläche des PAS.....	52
Aufbau der PN-Plattform mit PPVM.....	54
Kernelemente der PPVM und ihr Zusammenhang mit der CPI	55
Beispiel für Ausdrücke in BPMN-Modellen	57
Aufbau der PN-Plattform mit RPVM.....	63
Die durch die RPVM definierten Basisklassen für Ereignisse.....	64
Beispiel für ein Rete-Netzwerk.....	72
Aufbau der RPVM.....	77

Prozedurale und regelbasierte Zeiger zur Laufzeit.....	78
Modellelemente von DPIL.....	81
Ereignisse für die DPIL-Elemente	83
Lebenszyklus einer Aufgabeninstanz.....	84
Abstrakte Syntax für Regeln in DPIL	87
Abstrakte Syntax für Einschränkungen auf Objekten	88
Aufgaben der Sprachinfrastruktur.....	95
Zusammenfassung der Abdeckung von Prozessmustern durch DPIL und BPMN.....	112
Aufgabensicht bei der Ausführung des DECLARE-Beispiels nach dem Start, nach A und nach B	116
Strukturelle Bestandteile eines CMMN-Prozesses	117
Bestandteile von CMMN zur Modellierung von Regeln	118
Exemplarisches CMMN-Modell zur Überführung in DPIL.....	119
Hybride Modellierung des Aufnahmeprozesses einer psychiatrischen Klinik	123

1 Problemstellung und Zielsetzung

Der Erfolg einer Organisation – sei es ein produzierendes Unternehmen, ein Dienstleister, eine Forschungseinrichtung oder eine Klinik – hängt unter anderem von seiner Fähigkeit ab, seine Aufgaben auf strukturierte und zuverlässige Weise zu erfüllen. Eine Methode zur internen Strukturierung bildet das (Geschäfts-) Prozessmanagement¹. Der Einsatz der Methoden des Prozessmanagements kann durch unterschiedliche Motive und Ziele begründet sein:

Umsetzung regulatorischer Maßnahmen (engl. *compliance*): Als Reaktion auf die Finanzskandale großer amerikanischer Firmen wurde 2002 der *Sarbanes-Oxley Act* vom Senat der Vereinigten Staaten verabschiedet, um das Vertrauen der Anleger wiederherzustellen (von der Crone & Roth 2003). Dieses Rahmengesetz fordert eine nachvollziehbare Rechnungslegung und zudem die Existenz und die Einhaltung interner Kontrollmechanismen. Als Mittel der Wahl gilt hier ein prozessorientierter Ansatz (zur Muehlen & Ho 2006).

Etablierung des eigenen Qualitätsmanagements: Die Normenreihe EN ISO 9000 ff. bildet die Grundlage für Maßnahmen zum Qualitätsmanagement einer Organisation. Zu den Grundsätzen nach EN ISO 9001 gehört auch ein prozessorientierter Ansatz. Die Grundlage bildet der Deming-Kreis (auch PDCA-Zyklus, von engl. *plan-do-check-act*), der einen kontinuierlichen Verbesserungsprozess in die vier Schritte *Planen*, *Durchführen*, *Überprüfen* und *Handeln* gliedert (Deming 1982). Viele branchenspezifische Qualitätsnormen wie das VDA QMC Regelwerk in der Automobilindustrie oder das KTQ-Verfahren im Gesundheitswesen sind aus diesen Normen abgeleitet. Beim Export in die Vereinigten Staaten spielt vor allem das Rahmenwerk *Capability Maturity Model Integration* (CMMI) eine wichtige Rolle. Das CMMI fordert ausdrücklich einen prozessbasierten Ansatz (CMMI Product Team 2011).

Steigerung der Effizienz im Wettbewerb: Die Globalisierung und technischer Fortschritt zwingen (Unternehmens-) Organisationen ihr Produkt oder ihre Dienstleistung effizienter bereitzustellen als ihre Mitbewerber. Zur generellen Anhebung der Leistung einer Organisation ist das Prozessmanagement als wichtigstes organisatorisches Mittel anerkannt (Hammer & Champy 1993).

1.1 Methodik des Prozessmanagements

Die klassische Methodik des Prozessmanagements lässt sich in Form des in Abbildung 1.1 gezeigten Lebenszyklus beschreiben, den der Prozess als Artefakt durchläuft (Meerkamm 2011). Der Lebenszyklus gliedert sich in die fünf Phasen *Strategie*, *Entwurf*, *Modellierung*, *Ausführung* und *Überwachung* bzw. *Kontrolle*. Die vorliegende Arbeit befasst sich vor allem mit den Phasen der Modellierung und Ausführung eines Prozesses.

¹ Im Folgenden wird nicht weiter zwischen *Prozessen* und *Geschäftsprozessen* unterschieden.

1.1.1 Strategie und Entwurf

Die wesentlichen Ergebnisse der Phase des strategischen Managements sind die langfristigen Geschäftsziele. Prozessmanagementprojekte dienen anschließend zu deren Umsetzung. In dieser Phase werden die wichtigsten Prozesse – die Schlüsselprozesse (engl. *key processes*) – identifiziert und entsprechende Prozessverantwortliche (engl. *process owners*) ausgewählt (Weske 2012).

In der Entwurfsphase werden zunächst die formalen und inhaltlichen Anforderungen an die Prozesse selbst, an ihre Modellierung und an ihre Ausführung gesammelt. Außerdem wird definiert, wie diese Anforderungen umgesetzt werden sollen. Das Ergebnis ist die Entscheidung für eine *Prozessmodellierungssprache*, eine entsprechende *Plattform zur Ausführung* und eventuell weitere benötigte *Werkzeuge*.

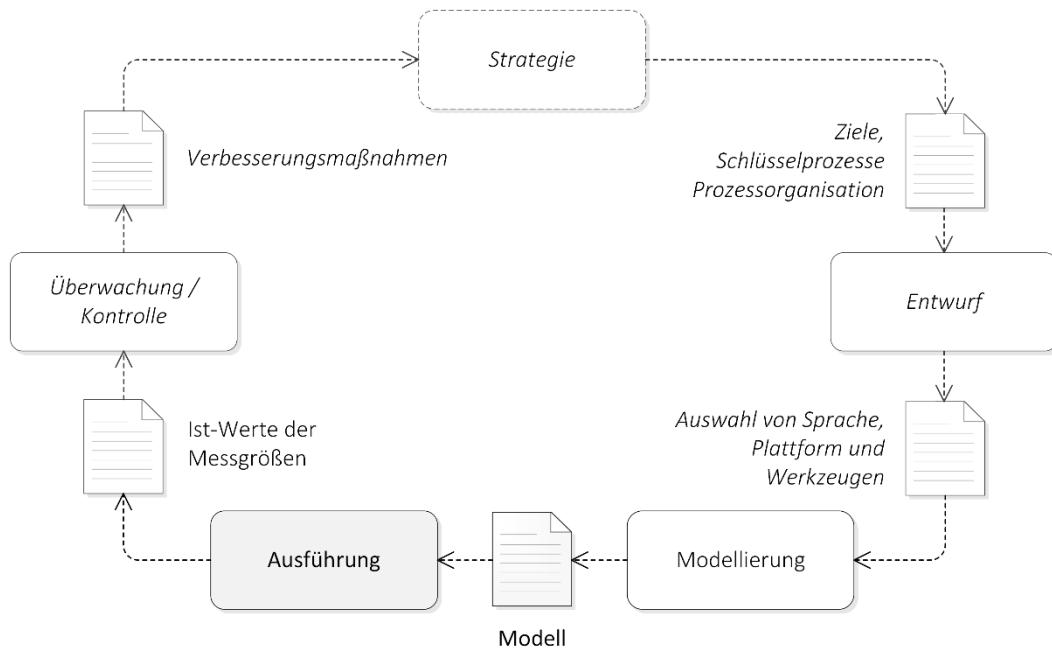


Abbildung 1.1 Konsolidierter Prozesslebenszyklus nach (Meerkamm 2011)

1.1.2 Modellierung von Prozessen

Die Phase der Modellierung beinhaltet die Erstellung eines *Prozessmodells*, also die Abbildung eines Prozesses der realen Welt (Meerkamm 2011). Zur Darstellung solcher Modelle wurden im Laufe der letzten 50 Jahre entsprechende Notationen entwickelt, von denen die meisten auf das Flussdiagramm zurückgehen (Fachnormenausschuß Informationsverarbeitung (FNI) im Deutschen Normenausschuß (DNA) 1966). Derartige Notationen werden im Folgenden (Prozess-) *Modellierungssprachen* genannt. Eine Modellierungssprache umfasst zunächst die Bausteine, aus denen ein Modell zusammengesetzt ist, sowie Regeln, nach denen diese Bausteine kombiniert werden müssen (*Syntax*). Außerdem umfasst eine Sprache die Bedeutung der einzelnen Bausteine sowie die Bedeutung von Kombinationen aus diesen Bausteinen (*Semantik*). Der dritte Aspekt einer Sprache umfasst die *Qualität* eines Modells, also Faktoren wie zum Beispiel die Wartbarkeit eines Prozessmodells (Bertoa & Vallecillo 2010).

Ein Flussdiagramm umfasst zum Beispiel die Symbole für den Beginn und das Ende des Prozesses (abgerundete Rechteck), einen Prozessschritt (Rechteck), eine Verzweigung (Raute) und den Fluss (Pfeil). Die Regeln für die Kombination dieser Elemente umfassen zum Beispiel, dass Prozessschritte mit dem jeweiligen Vorgang beschriftet werden müssen und dass eine Verzweigung über mindestens zwei ausgehende Flüsse verfügen muss. Die Elemente zusammen mit diesen Regeln bilden die Syntax von Flussdiagrammen. Die Semantik umfasst die Bedeutung des Modells, also seine Interpretation. Das in Abbildung 1.2 gezeigte Flussdiagramm beschreibt zum Beispiel einen Ablauf aus drei Schritten und einer Verzweigung. Zunächst werden die Eltern eines Patienten kontaktiert. Dabei wird die Entscheidung getroffen, ob die Eltern einer Aufnahme des Patienten zustimmen. Ist dies der Fall, dann wird der Patient der Klinik gemeldet. Lehnen die Eltern die Aufnahme ab, so wird der Vorbehandler des Patienten darüber informiert. Ein Qualitätskriterium von Flussdiagrammen ist zum Beispiel, dass der Fluss über das gesamte Diagramm hinweg von oben nach unten oder von links nach rechts verläuft. Dieses Kriterium ist im Beispiel erfüllt.

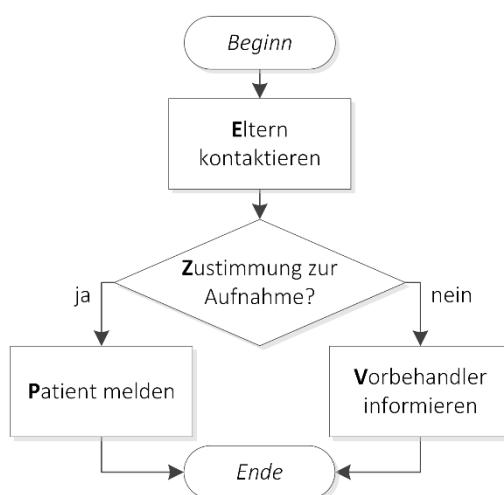


Abbildung 1.2 Exemplarisches Prozessmodell als Flussdiagramm

Ein Prozessmodell beschreibt letztlich eine Menge von möglichen alternativen Abläufen. Im Modell in Abbildung 1.2 ergeben sich durch die Verzweigung zwei mögliche vollständige Abläufe. Betrachtet man auch die unvollständigen Abläufe, also Zwischenzustände, dann ergeben sich insgesamt vier davon. Ein solcher Ablauf kann im einfachsten Fall als Liste von durchgeführten Prozessschritten notiert werden²:

- [] (noch kein Prozessschritt durchgeführt)
- [E]
- [E, P]
- [E, V]

Eine solche Liste wird auch als *Ereignisverlauf* (engl. *event log*) bezeichnet. Allgemein bezeichnet man mögliche Abläufe eines Prozesses – ob unvollständig oder nicht – als *Prozessinstanzen* des

² Der erste Buchstabe eines Prozessschritts dient als Abkürzung für diesen Schritt

Prozessmodells. Der Ereignisverlauf ist eine mögliche Darstellung einer Prozessinstanz (Workflow Management Coalition 1999).

1.1.3 Ausführung von Prozessen

Wurde ein Prozess der realen Welt in einem formalen Prozessmodell abgebildet, so lässt er sich durch ein IT-System unterstützen, also „ausführen“. Ein formales Modell ist in diesem Zusammenhang eines, das in einer Modellierungssprache mit unmissverständlicher und maschineninterpretierbarer Bedeutung (Semantik) ausgedrückt wird. Ein entsprechendes System wird mit jeweils leicht unterschiedlicher Bedeutung *Workflow Management System* (WfMS), *Workflow Engine*, *Workflow Enactment Service* (Workflow Management Coalition 1999) oder allgemein *Process-Aware Information System* (PAIS) (Reichert & Weber 2012) genannt. Um Verwechslungen zu vermeiden wird ein Software-System, das maschinenlesbare Prozessmodelle ausführt im Folgenden als *Prozessausführungssystem* (PAS) bezeichnet. Es

- interpretiert formale Prozessmodelle,
- verwaltet die Laufzeitinformationen zu Prozessinstanzen,
- weist personelle Aufgaben den entsprechenden Prozessteilnehmern zu,
- stellt prozessrelevante Daten den Prozessteilnehmern zur Verfügung,
- kommuniziert mit eingebundenen Diensten und Anwendungen (*Orchestrierung*)
- zeichnet den Verlauf von Prozessinstanzen auf (*Logging*) und
- erfasst relevante Kennzahlen und Messgrößen (*Monitoring*) (Reichert & Weber 2012).

Im Rahmen dieser Arbeit steht die Ausführung von Prozessmodellen in einem PAS im Vordergrund. Daneben sind aber weitere Arten der Verwendung eines Prozessmodells denkbar, zum Beispiel die Verwendung als Handbuch ohne Einsatz von IT (Meerkamm 2011).

1.1.4 Überwachung und Kontrolle

Um ein Qualitätsmanagement an die Ausführung anzuschließen, können die in einem PAS ausgeführten Prozesse überwacht werden. Das Hauptziel besteht darin, Soll- und Ist-Daten vergleichen zu können. Hierbei lassen sich zwei Ansätze unterscheiden (Meerkamm 2011):

- Überwachung (auch „Monitoring“): Bei gravierenden Abweichungen wird sofort in den Prozessablauf eingegriffen.
- Nachträgliche Kontrolle (auch „Controlling“): Durch eine Analyse anschließend zur Ausführung werden Konsequenzen für den Prozesslebenszyklus abgeleitet.

Die Erfassung von Kennzahlen während der Ausführung eines Prozesses und die Aufzeichnung von Prozessverläufen ermöglichen es einem PAS, beide Ansätze des Qualitätsmanagements zu unterstützen.

1.2 Prozessmanagement als Weiterentwicklung von Softwaresystemen

Die Verbreitung von Prozessmanagementsystemen wird als nächste Evolutionsstufe für die Architektur von Softwaresystemen gesehen. Die Entwicklung von frühen Softwaresystemen bis hin zu den PAS ist in Abbildung 1.3 gezeigt. Vor dem Aufkommen von Betriebssystemen wurde auf einem Computer jeweils immer nur ein Anwendungsprogramm zur selben Zeit ausgeführt. Die

Anwendung beinhaltete die Geschäftslogik, eine eigene Benutzeroberfläche, die Datenhaltung und die Verwaltung der Hardware und war damit nur auf einem bestimmten Computermodell lauffähig (Tanenbaum 2007) (1). Betriebssysteme wie OS/360 übernahmen ab Mitte der 60er Jahre die Verwaltung der Hardware und ermöglichten zudem die parallele Ausführung von Programmen und die Unterstützung ganzer Produktlinien (2). Der nächste wichtige Schritt war die Auslagerung der Datenhaltung durch die Entwicklung von Datenbankmanagementsystemen (DBMS) (3). Die Entwicklung von Programmbibliotheken für grafische Benutzeroberflächen und die Verbreitung des *Model View Controller* (MVC)-Musters (Gamma et al. 1994) verlagerte in der 80er Jahren einen weiteren Teil aus dem eigentlichen Anwendungsprogramm (4). Prozessmanagementsysteme ermöglichen es nun, die Geschäftsprozesse so auszulagern wie es zuvor mit der Datenhaltung und der Benutzerinteraktion geschehen ist (5) (van der Aalst 1996).

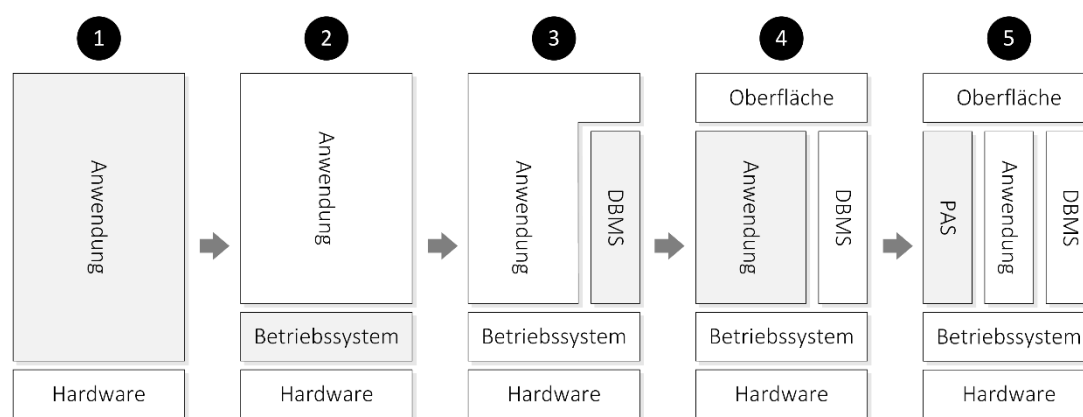


Abbildung 1.3 Historische Entwicklung von PAS (vgl. (van der Aalst 1996))

Die eigentliche Anwendung wird dann aus Prozessmodellen gebildet, die vom Prozessmanagementsystem interpretiert werden. Eine Änderung der Geschäftsabläufe erfordert somit lediglich eine Anpassung der Prozessmodelle. Die Übereinstimmung zwischen der Strategie, den Zielen und dem Bedarf einer Organisation und den Anforderungen an ein IT-System wird als *business/IT alignment* bezeichnet (Luftman 2005). Werden die Prozesse in einer Form modelliert, die für die IT- und die Fachabteilungen einer Organisation gleichermaßen zugänglich ist, dann lässt sich dadurch die semantische Lücke zwischen den beiden Welten verkleinern und das *business/IT alignment* verbessern.

1.3 Strikte und agile Prozesse

Bereits vor etwa 20 Jahren wurde erkannt, dass mindestens zwei grundlegende Arten von Prozessen unterschieden werden können (Jablonski 1994):

- wohlstrukturierte Routineprozesse, deren genauer Ablauf im Vordergrund steht und bereits zur Entwurfszeit vollständig festgelegt werden kann und muss (*strikte Prozesse*) und
- offene Prozesse, deren Ablauf sich nicht vollständig im Voraus festlegen lässt (*agile Prozesse*) (Reichert & Weber 2012)

Als einfache Beispiele führen Jablonski et al. den Prozess einer Reisekostenabrechnung an, dessen Ablauf genau festgelegt werden kann. Eine Verhandlung zwischen Menschen hingegen besteht zwar aus Vorschlägen, Fragen, Antworten, Zustimmung und Uneinigkeiten. Ihr genauer Ablauf kann allerdings im Voraus nicht festgelegt werden. Die Unterscheidung in strikte und agile Prozesse wird inzwischen auch im Bereich des Managements und der IT-Beratung anerkannt. Wie in diesem Bereich üblich wird die IT-Unterstützung von agilen Prozessen sogar als die völlig neue Methodik, nämlich als *Dynamic Case Management* (DCM) statt wie bisher als *Business Process Management* (BPM) gehandelt (Le Clair & Miers 2011).

Das Ziel dieser Arbeit besteht darin, sowohl strikte als auch agile Prozesse durch ein PAS umfassend zu unterstützen. In diesem Zusammenhang haben sich im Bereich der Prozessmodellierung zwei gegensätzliche Prinzipien herausgebildet. Jedes der beiden Prinzipien eignet sich für jeweils eine Art von Prozessen.

1.3.1 Strikte Prozesse und prozedurale Modelle

Die traditionellen Notationen zur Beschreibung von Prozessen wie Flussdiagramme (Fachnormenausschuß Informationsverarbeitung (FNI) im Deutschen Normenausschuß (DNA) 1966), EPKs (Software AG 2012) und BPMN (Object Management Group Inc. 2011) beruhen auf einer expliziten Angabe von sequentiellen, alternativen und parallelen Pfaden. Jeder mögliche Ablauf muss bereits zur Entwurfszeit bekannt sein (Reichert & Weber 2012). Dieses Prinzip existiert in identischer Form im Zusammenhang mit Programmiersprachen und wird dort prozedurale (oder imperative) Programmierung genannt. Der Begriff wurde auf die Prozessmodellierung übertragen und entsprechende Prozessmodelle als prozedurale Modelle bezeichnet (Dourish et al. 1996). Der Zusammenhang wird auch dadurch deutlich, dass das Flussdiagramm ursprünglich als „Programmablaufplan“ bekannt war (Fachnormenausschuß Informationsverarbeitung (FNI) im Deutschen Normenausschuß (DNA) 1966). Die prozedurale Modellierung ist also für strikte Prozesse besonders geeignet, bei denen die Abläufe zur Entwurfszeit bekannt sind und auch im Vordergrund stehen (Fahland et al. 2009). Hieraus ergibt sich Anforderung A1 an das im Rahmen dieser Arbeit entwickelte PAS:

A1 Strikte Prozesse müssen sich in einer prozeduralen Form modellieren lassen.

1.3.2 Agile Prozesse und regelbasierte Modelle

Bereits Mitte der 90er Jahre wurde erkannt, dass es Prozesse gibt, bei denen die genaue Abfolge der Schritte nicht oder nur zum Teil modelliert werden kann (Jablonski 1994). Solche Prozesse sind in starkem Maße abhängig von den Prozessteilnehmern, ihrem Expertenwissen und ihren Entscheidungen. Dieses Wissen lässt sich nicht oder nur teilweise erfassen und formalisieren. Daher spricht man auch von der Klasse der wissens- und entscheidungsintensiven Prozesse (engl. *knowledge- and decision-intensive processes*) (Vaculín et al. 2011). Der Forschungsbereich der Fallbehandlung (engl. *case management* oder *case handling*) bezeichnet sie als Fälle (engl. *cases*) (van der Aalst et al. 2005). Solche Prozesse erfordern also ein hohes Maß an „Flexibilität“ (Vaculín et al. 2011). Flexibilität kann in diesem Zusammenhang schlicht als ein Maß für die Anzahl der möglichen alternativen Abläufe aufgefasst werden. Ein Prozessmodell ist umso flexibler, je mehr alternative Prozessabläufe es ermöglicht (Schonenberg et al. 2008).

Um nun aber ein prozedurales Modell entsprechend flexibler zu gestalten, müssen die zusätzlichen Pfade (Alternativen) explizit eingefügt werden (Schonenberg et al. 2008). Sollen in einem Prozess zum Beispiel zwei Schritte nie zusammen durchgeführt werden, so kann der Modellierer dies in einem prozeduralen Prozess nicht direkt ausdrücken sondern muss mit Hilfe von exklusiven Verzweigungen (Entscheidungen) eine detaillierte Strategie modellieren, die diesen Sachverhalt implementiert (Maja Pešić et al. 2007). Diese Eigenschaft prozeduraler Modelle kann im Extremfall zu einer Überspezifikation führen (Reijers et al. 2013). Ein überspezifiziertes Modell ist restriktiver, als es die Sachverhalte eigentlich erfordern würden.

Prozeduralen Prozessmodellen stehen *regelbasierte* (auch „deklarative“) Modelle gegenüber. In einem regelbasierten Modell sind zunächst alle Ausführungspfade erlaubt. Ohne Einschränkungen im Prozessmodell muss es den Prozessteilnehmern also möglich sein, die modellierten Aktivitäten in beliebiger Reihenfolge durchzuführen und auf die Datenobjekte beliebig zuzugreifen. Je mehr Einschränkungen dem Modell hinzugefügt werden, desto weniger Szenarien werden ermöglicht. Um ein solches Modell flexibler zu gestalten, müssen also lediglich Einschränkungen entfernt oder abgeschwächt werden (Schonenberg et al. 2008). Zudem stehen in einem regelbasierten Modell nicht die Abfolgen der Schritte sondern eher übergreifende Zusammenhänge im Vordergrund (Fahland et al. 2009). Ein regelbasierter Ansatz eignet sich also eher zur Beschreibung von agilen Prozessen (Vaculín et al. 2011). Hieraus ergibt sich Anforderung A2 an das zu entwickelnde PAS:

A2 Agile Prozesse müssen sich in einer regelbasierten Form modellieren lassen.

Der regelbasierte Ansatz erlaubt es also, Modelle bereits zur Entwurfszeit flexibler zu gestalten (engl. *flexibility by design*). Dem gegenüber stehen Ansätze, die es erlauben, laufende prozedurale Prozessmodelle bei Bedarf zu ändern, um die Flexibilität des PAS zu erhöhen (engl. *flexibility by change*). Sowohl Prozessschritte als auch Verknüpfungen können hinzugefügt oder entfernt werden (Schonenberg et al. 2008). Hierzu müssen laufende Prozessinstanzen entweder abgebrochen oder ihr aktueller Zustand übertragen (migriert) werden. Außerdem erfordert eine Änderung immer auch tiefgehende Kenntnis des Prozessmodells. Die Gruppen der Modellierer und die der Prozessteilnehmer umfassen aber üblicherweise nicht dieselben Menschen. Die Prozessteilnehmer haben daher in den seltensten Fällen weitreichende Kenntnisse des zugrundeliegenden Prozessmodells. Zur Laufzeit änderbare Prozessmodelle stellen also alleine keine Lösung für die Unterstützung agiler Prozesse dar.

Daneben bieten regelbasierte Modelle einen weiteren entscheidenden Vorteil gegenüber der prozeduralen. Wird ein Prozessmodell verändert, so entsprechen die laufenden Instanzen des Prozesses, die vor dessen Änderung gestartet wurden, höchstwahrscheinlich nicht mehr diesem Prozess. Die laufenden Instanzen können also entweder weiterhin entsprechend der vorherigen Version des Prozesses durchgeführt, abgebrochen oder in die neue Version des Prozesses überführt, also „migriert“ werden. Diese Migration von Instanzen gestaltet sich bei regelbasierten Modellen einfacher als bei prozeduralen, weil im regelbasierten Fall im neuen Prozessmodell kein passender Zustand für die Instanz gefunden werden muss (M Pešić et al. 2007). Wird eine Prozessinstanz zum Beispiel als Ereignisverlauf repräsentiert, so kann zur Migration der bisherige

Verlauf, der gemäß dem bisherigen Modell entstanden ist, für das neue Modell übernommen werden und anschließend gemäß den neuen Prozessregeln geprüft werden.

1.3.3 *Verpflichtende und empfohlene Anteile in agilen Prozessen*

Ein Ziel der regelbasierten Modellierung ist es also, mehr Flexibilität durch weniger Modellierungskonstrukte zu ermöglichen. Mit Blick auf die entstehende Vielfalt an Alternativen ist es also besonders in agilen Prozessen sinnvoll zu unterscheiden, welche Abläufe verpflichtend (konstant) und welche lediglich empfohlen (veränderlich) sind (Regev & Wegmann 2005). Logisch gesehen müssen also zwei Modalitäten unterschieden werden können (Nonnengart & Ohlbach 1992). Dies ist durch die Praxis motiviert. So wird bei Prozessen im Gesundheitswesen zwischen klinischen Leitlinien (empfohlen) und Richtlinien (verpflichtend) unterschieden. Dies lässt sich auf eine Unterscheidung in gesetzlichen Rahmen und bewährtes Vorgehen (engl. *good practice*) verallgemeinern.

A3 Bei der Modellierung von regelbasierten Prozessen müssen verpflichtende und empfohlene Handlungen unterschieden werden können.

Eine Unterscheidung in verpflichtende und empfohlene Prozessbestandteile ist zunächst unabhängig vom Modellierungsprinzip und daher sowohl für prozedurale als auch für regelbasierte Modelle denkbar und sinnvoll. Dennoch bezieht sich die Anforderung in erster Linie auf regelbasierte Modelle und soll auch im Rahmen dieser Arbeit nur in diesem Bereich erfüllt werden. Dies liegt im Prinzip der Modellierung begründet. Abbildung 1.4 zeigt hierzu einen einfachen Zusammenhang in regelbasierter Notation (❶ und ❷) und in prozeduraler Notation (❸ und ❹):

- Der Artikel muss zunächst abgeholt werden, bevor er verpackt werden kann.
- Die Rechnung kann jederzeit vorbereitet werden.

Im regelbasierten Modell lässt sich dies als Einschränkung durch ein einziges Konstrukt realisieren, nämlich durch die Abhängigkeit zwischen „Artikel abholen“ und „Artikel verpacken“ (Linie und Diamant). Dieses Konstrukt lässt sich nun als empfohlen (❶, gestrichelte Linie) oder verpflichtend (❷, durchgezogene Linie) kennzeichnen.

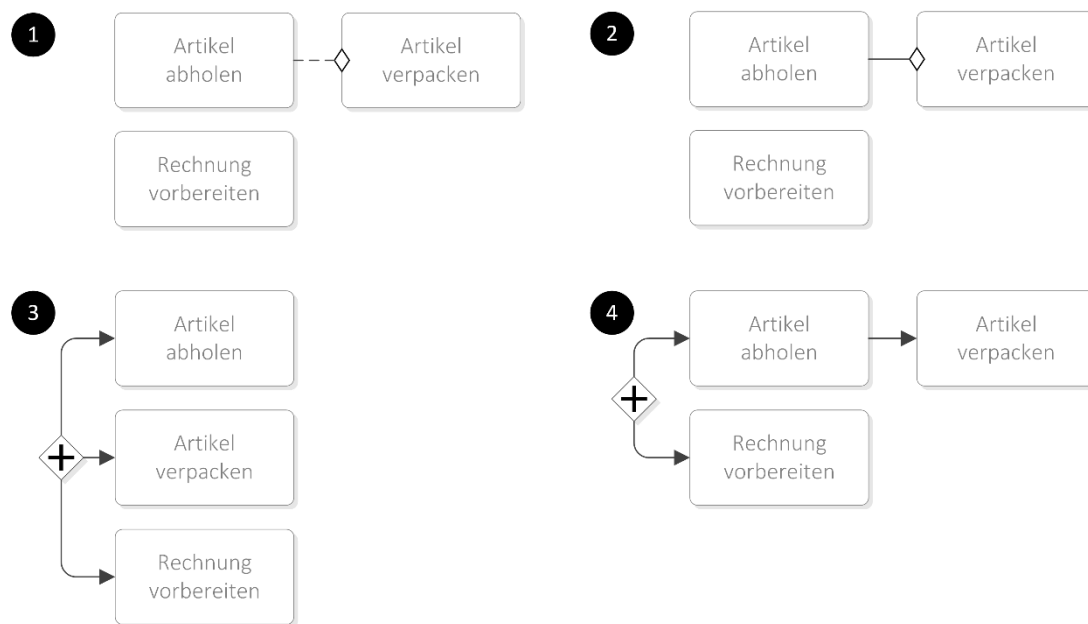


Abbildung 1.4 Exemplarischer Prozess in prozeduraler und regelbasierter Notation

Das prozedurale Modell hingegen muss alle möglichen Abläufe explizit enthalten. Modell **3** stellt also alle möglichen Abläufe dar. Alle drei Aktivitäten sind durch eine parallele Verzweigung (Raute mit Plus) voneinander entkoppelt, können also in beliebiger Reihenfolge und nebenläufig zueinander durchgeführt werden. Modell **4** stellt die empfohlenen Abläufe dar. Die beiden Aktivitäten „Artikel abholen“ und „Artikel verpacken“ sind durch einen Fluss (Pfeil) verbunden, werden also nacheinander durchgeführt. Die dritte Aktivität ist durch eine parallele Verzweigung von den beiden anderen entkoppelt, kann also jederzeit und nebenläufig zu den anderen durchgeführt werden. Modell **4** ist für sich äquivalent zu Modell **2**. Derzeit ist nicht klar, wie sich im Fall einer prozeduralen Notation die beiden Modelle vereinen und wie im regelbasierten Fall durch eine einfache Markierung unterscheiden lassen können.

1.3.4 Entscheidungsunterstützung

Anders als bei strikten Prozessen wird im Rahmen von agilen Prozessen der Großteil der Entscheidung erst zur Laufzeit des Prozesses und nicht im Voraus zu dessen Modellierung getroffen. Die Steuerung kann also umso weniger durch das Prozessmodell und damit durch das PAS erfolgen. Ein Großteil der Steuerung erfolgt durch die Prozessteilnehmer, wobei das PAS sie unterstützt, so wie auch ein Navigationssystem einen Fahrer unterstützt. Ein Navigationssystem berechnet die Route fortwährend neu, das heißt sie ist nicht festgelegt sondern ändert sich dem Verhalten des Fahrers und eventuell weiteren Einflüssen entsprechend (van der Aalst 2009). Ein dieser Analogie entsprechendes PAS ist also weniger ein steuerndes Automatisierungswerkzeug sondern vielmehr ein Entscheidungsunterstützungssystem (EUS), bei dem die Benutzer stets die Kontrolle behalten. Eine zentrale Eigenschaft eines solchen Systems ist, dass es seine Ausgaben erklärt und qualifiziert (Turban et al. 2010). Ein Navigationssystem kann eine alternative Route empfehlen, weil sie schneller zum Ziel führt als die aktuell verfolgte. Bezogen auf die Prozessausführung bedeutet das, dass vorgeschlagene Handlungen empfohlen sein können und

sich diese Empfehlung auf das Prozessmodell und den aktuellen Ausführungskontext zurückführen lässt.

A4 Die Empfehlung oder das Abraten von Handlungen muss sich während der Ausführung agiler Prozesse auf das Prozessmodell und den Ausführungskontext zurückführen lassen.

1.3.5 Modellierungsperspektiven und Änderung von Sprachen

Unabhängig vom zugrundeliegenden Modellierungsprinzip (prozedural oder regelbasiert) wurden zur Strukturierung von Prozessen grundlegende Perspektiven identifiziert (Curtis 1992) und auf durch IT-Systeme ausführbare Prozessmodelle übertragen (Jablonski 1994):

Funktion: Die Funktionsperspektive beschreibt die funktionalen Bausteine eines Prozesses. Man unterscheidet dabei elementare und zusammengesetzte Prozessschritte. Elementare Schritte bilden die kleinsten Arbeitseinheiten und erfordern üblicherweise personelle oder technische Ressourcen. Zusammengesetzte Prozessschritte hingegen verweisen auf ein untergeordnetes Prozessmodell, bestehen also wiederum aus Prozessschritten. Sie ermöglichen die Wiederverwendung von Teilprozessen und die Strukturierung von umfangreichen Prozessmodellen (Curtis 1992) (Reichert & Weber 2012). So setzt sich im Prozessmodell einer medizinischen Operation der Prozessschritt „Narkose einleiten“ zum Beispiel aus den Schritten „Venenkanüle anlegen“, „Monitoring anschließen“, „Sauerstoff verabreichen“, „Analgetikum injizieren“ und „Hypnotikum injizieren“ zusammen.

Verhalten: Die Verhaltensperspektive beschreibt das zeitliche Verhalten eines Prozesses und wird in klassischen Prozessmodellierungssprachen durch den Kontrollfluss abgedeckt. Der Kontrollfluss bestimmt die Reihenfolge in der die Schritte durchgeführt werden und eventuelle Randbedingungen.

Organisation: Die Organisationsperspektive beschreibt die Zuweisung von personellen Prozessschritten zu Prozessteilnehmern. Zu diesem Zweck referenzieren Prozesse üblicherweise ein organisatorisches Modell, das Personen, organisatorische Einheiten und Beziehungen abbildet (Bußler 1998).

Information: Die Informationsperspektive beschreibt welche Informationseinheiten an welcher Stelle im Prozess erzeugt oder manipuliert werden. Hier werden üblicherweise drei Kategorien von Informationen unterschieden (Workflow Management Coalition 1999):

- *Anwendungsdaten* werden von einer in den Prozess eingebundenen Anwendung außerhalb des WfMS verwaltet.
- *Prozessrelevante Daten* beeinflussen den Verlauf einer Prozessinstanz, also Zustandsübergänge oder organisatorische Zuweisungen.
- *Prozesskontrolldaten* sind für das WfMS interne Daten, die zum Beispiel Zustandsinformationen beinhalten und für die eingebundenen Anwendungen nicht zugänglich sind.

In (Jablonski & Bußler 1996) wird darüber hinaus eine weitere grundlegende Perspektive definiert:

Operation: Die Operationsperspektive beschreibt, wie ein elementarer Prozessschritt implementiert ist, das heißt was genau zu tun ist, wenn dieser Schritt ausgeführt wird. Üblicherweise wird die Implementierung eines Schritts als *black box* behandelt, das heißt die Abfolge von Schritten wird unabhängig von deren Implementierung koordiniert (Reichert & Weber 2012).

Diese Perspektiven wurden im Rahmen des *MOBILE*-Projekts erstmals alle in einem WfMS umgesetzt (Jablonski 1994) und finden sich heute in allen bedeutenden Modellierungsansätzen wieder (Scheer et al. 2005; Object Management Group Inc. 2011).

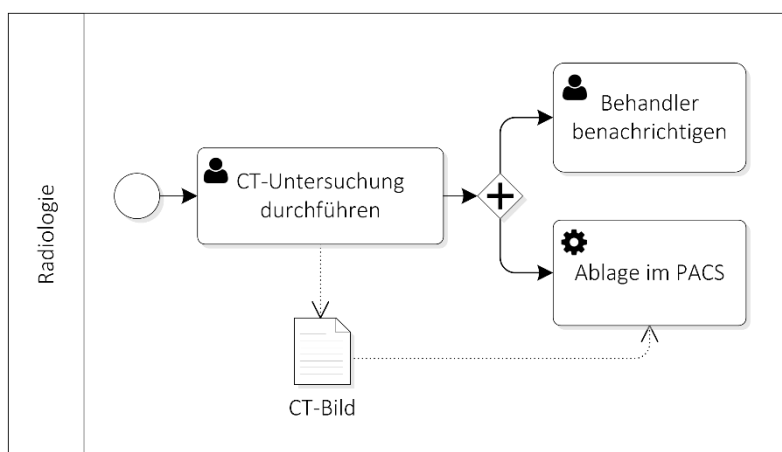


Abbildung 1.5 Prozessmodell in der BPMN-Notation mit den fünf grundlegenden Perspektiven

Abbildung 1.5 zeigt ein Prozessmodell in der BPMN-Notation (Object Management Group Inc. 2011). Es enthält

- den Prozess und Aktivitäten als Elemente der Funktionsperspektive,
- Ereignisse (Kreis), Verzweigungen (Rauten) und Sequenzflüsse (durchgezogene Pfeile) als Elemente der Verhaltensperspektive,
- die Bande (engl. *lane*) als Element der Organisationsperspektive,
- Datenobjekte (Dokumentsymbol) und -verbindungen (gepunktete Pfeile) als Elemente der Informationsperspektive und
- die verschiedenen Aufgabentypen als Element der Operationsperspektive. Es wird zum Beispiel zwischen personellen Aufgaben (Menschsymbols) und automatischen (Zahnradsymbol) unterschieden.

Sowohl im Bereich der Forschung als auch bei der Standardisierung gelten die oben eingeführten grundlegenden Perspektiven also als anerkannt. Das entwickelte PAS muss also in der Lage sein, alle fünf Perspektiven zu interpretieren.

A5 Die unterstützten Prozessmodelle und das Ausführungssystem müssen die fünf grundlegenden Perspektiven abdecken.

Besonders in agilen Prozessen stehen übergreifende Zusammenhänge im Vordergrund. Entsprechende Prozessmodelle umfassen oft Konstrukte, die sich gleichzeitig auf mehrere

Perspektiven beziehen (Iglar, Faerber, et al. 2010). So könnte ein Prozessmodell zum Beispiel fordern, dass ein Dokument – wenn es von einem Auszubildenden erstellt wurde – vor seiner Veröffentlichung von dessen Vorgesetzten geprüft werden muss. Der Ablauf hängt in diesem Fall also von der Informations- und Organisationsperspektive des Prozesses ab. Fasst man ein Modellierungskonstrukt als Implikation („wenn-dann-Regel“) auf, dann bedeutet eine perspektivenübergreifende Modellierung, dass sich sowohl die Bedingung(en) der Regel als auch ihre Konsequenz(en) auf alle von der Modellierungssprache abgedeckten Perspektiven beziehen kann. Im Beispiel führt also ein informationsbezogener und zugleich organisatorischer Zusammenhang zu einer verhaltensbezogenen und zugleich organisatorischen Einschränkung. Somit ergibt sich Anforderung A6.

A6 Agile Prozesse müssen sich perspektivenübergreifend modellieren lassen.

Eine perspektivenübergreifende und zugleich prozedurale Modellierung ist denkbar und sinnvoll, dennoch bezieht sich Anforderung A6 in erster Linie auf agile Prozesse und damit auf regelbasierte Modelle. Dies liegt darin begründet, dass aktuelle Vertreter der prozeduralen Modellierung wie die BPMN einen konsistenten Kontrollfluss voraussetzen und alle weiteren Perspektiven diesem unterordnen und getrennt voneinander behandeln (Object Management Group Inc. 2011). Eine perspektivenübergreifende Modellierung im Sinne der Anforderung ist somit nur eingeschränkt möglich. Jede Anforderung muss zunächst in ihre jeweiligen Perspektiven zerlegt werden, bevor sie abgebildet werden kann.

Die Vollständigkeit der Modellierungssprache in Bezug auf die Perspektiven hängt von ihrem jeweiligen Einsatzgebiet ab (Curtis 1992). So wurden neben den grundlegenden Perspektiven im Laufe der Zeit weitere vorgeschlagen. Die *Zielperspektive* (Talib et al. 2010) dient zum Beispiel der Definition von Leistungsindikatoren auf deren Basis das Personal in Produktionsprozessen effizienter eingesetzt werden kann. Die *historische Perspektive* behandelt zum Beispiel die Aufzeichnung aller relevanten Daten einer Workflow-Anwendung „um für statistische Analysen zur Laufzeit, Auswertungen zur Prozessoptimierung oder betrieblichen Revisionsanforderungen abgelegt zu werden“ (Schludt 2004). Eine Modellierungssprache kann sich aber nicht nur durch domänenspezifische Erweiterungen sondern auch in ihrem Kern verändern. Abbildung 1.6 zeigt die Entwicklung wichtiger Prozessmodellierungsstandards über zehn Jahre hinweg und nach Standardisierungsgremien gruppiert.

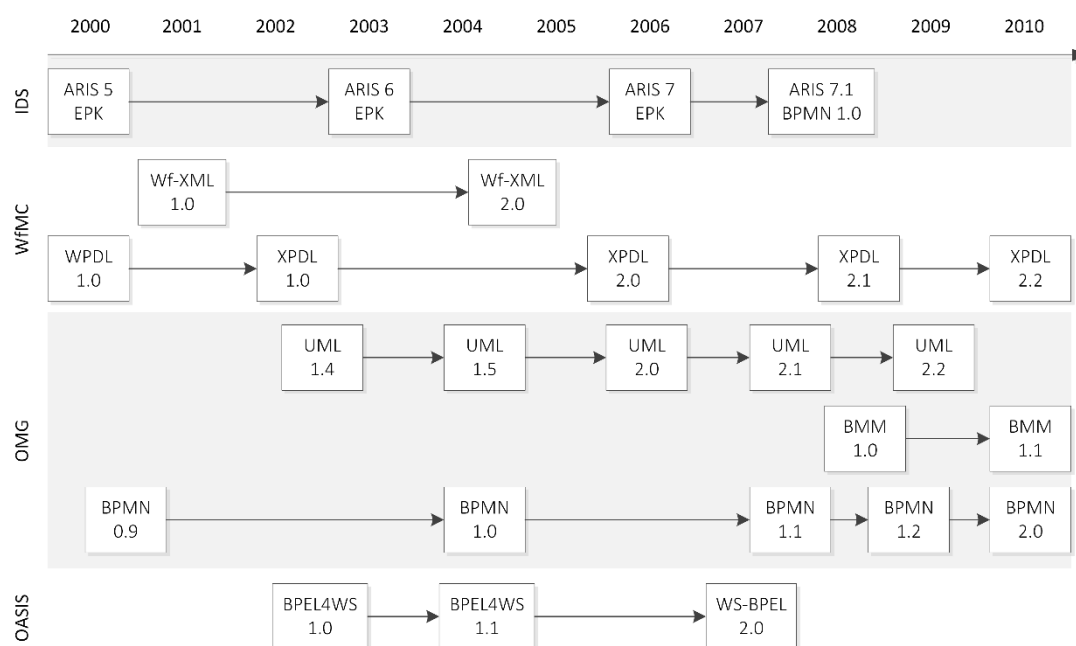


Abbildung 1.6 Historische Entwicklung wichtiger Standards im Prozessmanagement nach (Bartonitz 2009)

Man erkennt in Abbildung 1.6, dass die relevanten Modellierungsstandards in regelmäßigen Abständen verändert werden und damit üblicherweise entsprechende Änderungen an den Modellierungs- und Ausführungssystemen notwendig sind. Diese Änderungen ziehen allerdings umfangreiche Softwareentwicklungsprojekte und entsprechenden Wartungsaufwand nach sich, woraus sich Anforderung A7 ergibt:

A7 Änderungen am Kern der Sprache sowie domänenspezifische Erweiterungen sollen möglichst keine Änderungen am Kern des Ausführungssystems nach sich ziehen.

1.3.6 Mischung beider Prozessarten

Der Gesamtprozess einer Organisation besteht üblicherweise sowohl aus tendenziell festgelegten als auch eher dynamischen Teilprozessen. In Abbildung 1.7 ist der Prozess der Patientenaufnahme einer Station für Kinder- und Jugendpsychiatrie als BPMN-Diagramm dargestellt. Den Rahmen bildet ein festgelegter Prozess. Sobald ein Bett der Station frei wird, wird ein Patient aus der Warteliste entnommen und seine Eltern werden kontaktiert. Stimmen beide Elternteile der Aufnahme zu, wird ein Termin vereinbart. Stimmen sie nicht zu, wird der Vorbehandler informiert. Bestimmte Anteile lassen sich allerdings nicht vollständig spezifizieren. Die Aufnahme durch die Pflege und die Schritte nach der Anamnese durch den Arzt lassen sich nicht im Detail vorgeben³.

³ Die als Anmerkung eingetragenen Empfehlungen lassen sich nicht formal im Modell verankern. Die Darstellung als BPMN-Diagramm bietet daher nicht die Möglichkeit sowohl prozedurale als auch regelbasierte Anteile im selben Modell zu notieren.

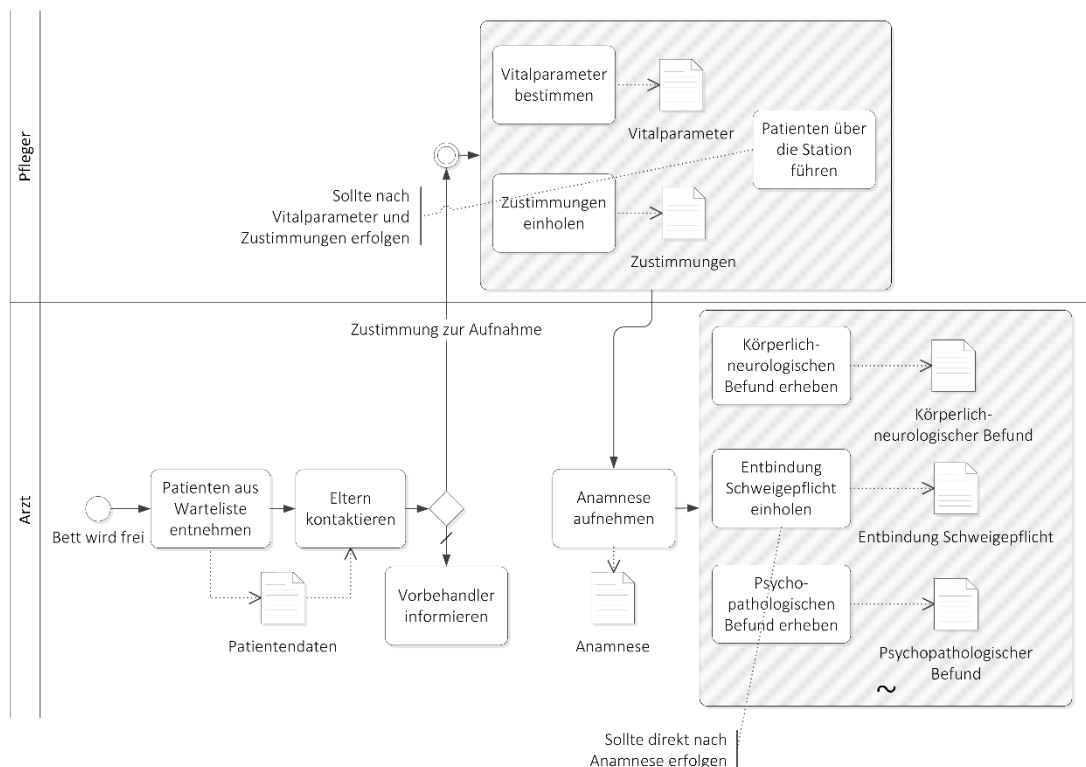


Abbildung 1.7 Vereinfachter Aufnahmeprozess einer psychiatrischen Station aus strikten und agilen Anteilen

Um derartige gemischte Prozesslandschaften unterstützen zu können, muss ein Ausführungssystem sowohl prozedurale als auch regelbasierte Prozessmodelle integriert ausführen können (Gronau & Weber 2004).

A8 Ein Ausführungssystem muss sowohl prozedurale als auch regelbasierte Prozessmodelle integriert ausführen können.

Unter einer integrierten Ausführung ist zu verstehen, dass der Wechsel zwischen Teilprozessen unterschiedlicher Modellierungsprinzipien nach außen hin transparent geschieht, also von den Prozessteilnehmern und eventuell angebundenen Systemen nicht bemerkt wird. Hinzu kommt, dass Konzepte, die für eine Hierarchie von Modellen desselben Modellierungsprinzips erlaubt sind, auch für eine Hierarchie unterschiedlicher Modelle möglich sein müssen. Dies betrifft den Geltungsbereich von Datenobjekten und die Wiederverwendung von Teilprozessen.

Voraussetzung für die integrierte Ausführung kombinierter Prozessmodelle ist eine ebenso integrierte Modellierung. Die jeweiligen Sprachen müssen Mechanismen bereitstellen, die sowohl eine Einbettung von „fremden“ Teilprozessen ermöglichen, als auch eine Einbettung ihrer Prozesse in „fremde“ Modelle. Entsprechende Modellierungsumgebungen müssen diese Mechanismen umsetzen.

1.4 Methodik und Zielsetzung

Die vorliegende Arbeit folgt der Forschungsmethodik des *Design Science*. Diese Methodik erweitert die Fähigkeiten von Menschen und Organisationen, indem sie ein innovatives und

zweckmäßiges Artefakt für einen relevanten Problembereich erschafft. Das Artefakt muss einen Nutzen für das Problem bringen, was im Rahmen einer sorgfältigen Evaluation gezeigt werden muss. Ebenfalls entscheidend ist die Neuartigkeit des Artefakts. Es muss also ein bisher ungelöstes Problem lösen oder ein bestehendes Problem auf effektivere oder effizientere Art lösen als es bisher der Fall ist. Das Artefakt selbst muss klar definiert, formal beschrieben, kohärent und in sich konsistent sein. Der Prozess seiner Erstellung und oft auch das Artefakt selbst umfassen oder ermöglichen einen Suchprozess. Während dieses Prozesses wird ein Problemraum konstruiert und ein Mechanismus vorgestellt oder vorgeführt, mit dem eine effektive Lösung gefunden wird. Schließlich müssen die Ergebnisse dieser Forschung veröffentlicht werden. Zum einen muss dabei die technische Gemeinschaft adressiert werden, also Wissenschaftler, die die Forschung erweitern, und Anwender, die sie umsetzen. Zum anderen muss die Forschung der betriebswirtschaftlichen Gemeinschaft zugänglich gemacht werden, also Wissenschaftlern, die sie im Zusammenhang untersuchen, und Anwender, die sie im eigenen Unternehmen einsetzen. Die Methode des Design Science steht im Gegensatz zur Verhaltensforschung, die Theorien zur Erklärung oder zur Vorhersage des Verhaltens von Menschen oder Organisationen entwickelt und verifiziert (Hevner et al. 2004).

Es wird oben gezeigt, dass das Prozessmanagement ein anerkanntes Mittel zur Strukturierung von Organisationen ist und sich die Prozesse einer Organisation aus strikten und agilen zusammensetzen. Im Verlauf der Arbeit wird außerdem gezeigt, dass bestehende Ansätze diese Anforderung nur unzureichend erfüllen können. Es wird daher als Lösungsvorschlag ein innovatives Artefakt – der Prototyp eines Prozessausführungssystems – entwickelt. Der Zweck dieses Systems ist es, das genannte Problem zu lösen, also eine umfassende IT-Unterstützung der Prozesse einer Organisation zu ermöglichen. Dazu muss es prozedurale Modelle strikter Prozesse (A1), regelbasierte Modelle agiler Prozesse (A2) und Kombinationen beider (A8) integriert ausführen können. Dabei werden alle fünf grundlegenden Perspektiven der Prozessbeschreibung berücksichtigt (A5). Vor allem agile Prozesse müssen perspektivenübergreifend (A6) und modal (A3) modelliert werden können. Das System muss bei der Ausführung dynamischer Prozesse seine Empfehlungen anhand des Prozessmodells und falls nötig auch anhand des Ausführungskontexts erklären können (A4). Änderungen und/oder Erweiterung an der Modellierungssprache sollen sich möglichst nicht auf das Ausführungssystem auswirken (A7). Der Prototyp wird mit Hilfe anerkannter Modellierungsansätze spezifiziert, konstruiert und unmissverständlich dokumentiert. Eine Evaluation auf der Basis eines etablierten Rahmenwerks zeigt im Verlauf der Arbeit, dass der Prototyp tatsächlich in der Lage ist, das Problem zu lösen.

1.5 Lösungsskizze und Beitrag

Der Schlüssel zur Beherrschung der Komplexität von Computersystemen ist die Einteilung in Abstraktionsebenen, die durch wohldefinierte Schnittstellen voneinander getrennt sind. Die *virtuelle Maschine* (VM) ist ein Konzept, das eine solche Einteilung ermöglicht (Smith & Ravi 2005). Das Prinzip wurde ursprünglich auf Sprachebene entwickelt, um vom Betriebssystem zu abstrahieren und somit höhere Programmiersprachen von der Plattform zu entkoppeln. So wird zum Beispiel die Sprache *Java* in die Zwischensprache *Java Bytecode* übersetzt, die dann von der *Java Virtual Machine* (JVM) interpretiert wird. Implementierungen der JVM existieren für die

verschiedensten Architekturen und Betriebssysteme, womit Java praktisch plattformunabhängig wird. Obwohl die JVM ursprünglich nur für die Ausführung kompilierter Java-Programme entworfen wurde, existiert mittlerweile eine Vielzahl an Compilern, mit denen sich Sprachen wie Pascal, PHP, Python oder JavaScript auf der JVM ausführen lassen. Eine VM führt also eine ganze Familie an Sprachen aus, solange sie sich auf die Zwischensprache der VM abbilden lassen. Ein weiteres Beispiel für dieses Muster ist der Warren-Automat (engl. *Warren Abstract Machine*, kurz WAM) für die Ausführung der Sprache Prolog (Warren 1983).

1.5.1 Virtuelle Maschine zur Ausführung von Prozessmodellen

Das Konzept der virtuellen Maschine lässt sich wie in (Baeyens & Faura 2007) gezeigt auf die Ausführung von Prozessmodellen übertragen. Bestehende prozedurale Prozessmodellierungssprachen werden auf eine gemeinsame Grundlage zurückgeführt. In diesem Fall haben alle prozeduralen Sprachen gemein, dass sie ein Netz aus Knoten und Kanten darstellen. Nur das Verhalten der Knoten ist für jede Sprache spezifisch. Die virtuelle Maschine realisiert zentrale Aspekte, die nicht für jede Modellierungssprache erneut implementiert werden müssen:

Persistenz: Die dauerhafte Speicherung von prozessrelevanten Daten, Prozesskontrolldaten und historischen Daten ist zunächst nicht zwingend notwendig. Wird sie jedoch benötigt, so kann sie durch die virtuelle Maschine einheitlich und transparent abgedeckt werden. Auch der transaktionale Zugriff auf die Daten kann dabei sichergestellt werden.

Prozessvariablen: Der Zugriff auf Informationen, die spezifisch für eine Prozessinstanz sind, und ihr Geltungsbereich kann durch die virtuelle Maschine vereinheitlicht werden.

Dekomposition und Nebenläufigkeit: Die unabhängige Ausführung von nebenläufigen Pfaden kann einheitlich und transparent abgedeckt werden. Auch die Verzweigung in einen Subprozess stellt einen nebenläufigen Pfad dar, so dass die Dekomposition von Prozessen auch Aufgabe der virtuellen Maschine ist.

Ausführungssemantik: Die virtuelle Maschine nimmt eine gemeinsame Grundlage für die Modellierungssprachen an und damit auch eine gemeinsame grundlegende Ausführungssemantik. Im Fall der PVM ist dies die Vereinheitlichung zu einem Netz aus Knoten und Kanten, das von Zeigern durchlaufen wird. Diese Gemeinsamkeiten können bereits durch die virtuelle Maschine zuverlässig realisiert werden und müssen für eine konkrete Modellierungssprache nur konfiguriert werden.

Operationen: Über das in (Baeyens & Faura 2007) gezeigte Konzept hinaus sollen in der vorliegenden Arbeit auch die Operationen eines Prozesses durch die virtuelle Maschine vereinheitlicht werden. Die Implementierung elementarer Prozessschritte soll also unabhängig von der Modellierungssprache abgedeckt werden. Hierzu zählen:

- Die Verwaltung von Aufgaben, die aus personellen Prozessschritten (engl. *human tasks*) hervorgehen.
- Die Kommunikation mit externen Diensten, die im Prozess eingebunden werden.
- Die Ausführung von Skripten, die in den Prozess eingebettet sind.

Die virtuelle Maschine realisiert also alle zentralen Eigenschaften eines Ausführungssystems unabhängig von der Prozessmodellierungssprache. Ändert sich eine von der VM unterstützte Modellierungssprache, so muss nicht das Ausführungssystem sondern lediglich die Abbildung auf die Zwischensprache geändert werden. Entsprechend Anforderung A7 kann damit der Aufwand für Entwurf, Implementierung und Wartung von Prozessmodellierungssprachen deutlich gesenkt werden.

1.5.2 Ausführung prozeduraler und regelbasierter Prozessmodelle

Die Bedeutung (Semantik) prozeduraler Prozessmodellierungssprachen wird üblicherweise mit Hilfe von Marken (engl. *tokens*) beschrieben, die entlang des Kontrollflusses die Elemente des Prozesses durchlaufen. Dabei wird für jedes Modellelement beschrieben, wie es mit einer Marke interagiert. So wird zum Beispiel für eine Aktivität definiert, dass sie durchgeführt wird sobald sie von einer Marke erreicht wird und im Falle eines ausgehenden Flusses wiederum eine Marke generiert sobald sie abgeschlossen wurde. Die Simulation dieser sich bewegenden Marken bildet zugleich eine übliche und effiziente Methode zur Ausführung prozeduraler Prozessmodelle (Gfeller et al. 2011). Der Zustand ergibt sich maßgeblich aus den aktuellen Positionen der Marken im Netz. Das Beispiel in Abbildung 1.8 zeigt zwei Ausführungsschritte eines exemplarischen Modells. Nachdem die Marke (Dreieck) zuletzt auf Aktivität A stand, wandert es beim Wechsel auf den nächsten Zustand auf die darauf folgende parallele Verzweigung (Raute mit Plus). Diese erzeugt bei ihrer Aktivierung wiederum für jeden ausgehenden Fluss eine Marke. Diese Marken wandern zu den Aktivitäten B und C und aktivieren diese parallel (Object Management Group Inc. 2011).

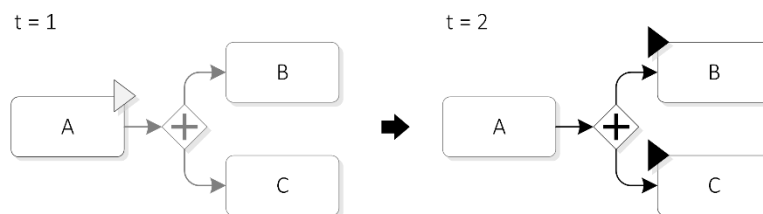


Abbildung 1.8 Ausführung prozeduraler Prozessmodelle

Alle untersuchten regelbasierten Konzepte basieren hingegen auf anderen Prinzipien zur Repräsentation des Zustands (Pešić 2006) (Iglar 2011) (Goedertier & Vanthienen 2007). Der Zustand eines solchen regelbasierten Modells lässt sich nicht in Form von Marken darstellen, sondern erfordert zum Beispiel einen vollständigen Ereignisverlauf der Prozessinstanz. Ein System, das zur effizienten Ausführung prozeduraler Modelle die Simulation von Marken implementiert, kann somit nicht zugleich in der Lage sein, solche regelbasierten Modelle auszuführen. Viele Aspekte der beiden Welten, wie die Datenhaltung oder die Realisierung der einzelnen Prozessschritte, überlagern sich. Dennoch muss ein System zur integrierten Ausführung prozeduraler und regelbasierter Modelle was die grundlegende Ausführungssemantik angeht spezialisierte Anteile für die Ausführung von jeweils prozeduralen und regelbasierten Modellen umfassen.

1.5.3 Prozedurale Prozess-VM als Basis für prozedurale Prozessmodellierungssprachen

Die *Process Virtual Machine* (PVM) nach Baeyens und Faura beschreibt ein allgemeines Modell für den Entwurf prozeduraler Ausführungssysteme. Es soll die gemeinsame Grundlage sowohl für Sprachen wie BPMN, BPEL und XPDL als auch domänenspezifischer Sprachen bilden und dabei die Kosten für Wartung, Entwurf und Implementierung entsprechender Ausführungssysteme reduzieren (Baeyens & Faura 2007). Das Konzept bildet die Grundlage für die *Procedural Process Virtual Machine* (PPVM), also diejenige Komponente des entwickelten PAS, die zur Ausführung prozeduraler Modelle dient.

1.5.4 BPMN als prozedurale Referenzsprache

Die *Business Process Model and Notation* (BPMN) hat sich als Standard zur Modellierung von prozeduralen Prozessmodellen etabliert und löst andere verbreitete Notationen wie die *Ereignisgesteuerten Prozessketten* (EPK) mehr und mehr ab (Decker et al. 2009). BPMN kombiniert eine grafische Darstellung mit einem umfangreichen Metamodell zum Transport ausführungsrelevanter Informationen. Die BPMN soll im Rahmen dieser Arbeit als Referenzsprache für prozedurale Prozessmodelle dienen. Die Ausführung von BPMN-Modellen auf der PPVM erfüllt somit Anforderung A1.

1.5.5 Regelbasierte Prozess-VM als Basis für regelbasierte Prozessmodellierungssprachen

Das im Rahmen dieser Arbeit entwickelte Prinzip der regelbasierten Prozess-VM (engl. *Rule-based Process Virtual Machine*, kurz *RPVM*) ist in Abbildung 1.9 gezeigt. Bestimmte Elemente eines regelbasierten Modells durchlaufen einen Lebenszyklus aus Ereignissen, der durch die VM verwaltet wird. Eine personelle Aufgabe zum Beispiel kann begonnen und abgeschlossen werden, während ein Datenobjekt gelesen und manipuliert werden kann. Der aktuelle Zustand des Prozesses ergibt sich dann aus dem bisherigen Verlauf an Ereignissen. Jede Aktion im Prozess, wie zum Beispiel das Abschließen einer Aktivität, führt zu einem neuen Ereignis im Verlauf (❶) und treibt gleichzeitig die Ausführung des Prozess weiter. Hierzu werden zunächst alle möglichen Ereignisse für den nächsten Zeitschritt generiert (❷). Zusammen mit dem bisherigen Verlauf ergeben sich dadurch jeweils mögliche Verläufe für die Prozessinstanz nach dem nächsten Zeitschritt. Diese möglichen Verläufe werden dann auf der Basis der Prozessregeln bewertet (❸). Die Bewertung kategorisiert die möglichen Verläufe (und damit die möglichen nächsten Ereignisse) als *nicht erlaubt*, *nicht empfohlen*, *neutral* oder *empfohlen*. Jedes Ereignis entspricht einer Aktion im Prozess. Das mögliche Ereignis *Aufgabe durch Teilnehmer Alice starten* entspricht so zum Beispiel der Zuweisung der Aufgabe zum Teilnehmer Alice. Alle nach der Bewertung erlaubten Ereignisse werden also entsprechend interpretiert und führen zu den jeweiligen Aktionen im Prozess (❹). Personelle Aufgaben werden zugewiesen oder entzogen, Dienste aufgerufen oder der Zugriff auf Datenobjekte erlaubt oder verwehrt. Die jeweilige Bewertung einer Aktion als nicht empfohlen, neutral oder empfohlen kann dabei durch die Teilnehmer eingesehen werden. Empfohlene beziehungsweise nicht empfohlene Aktion können auf die jeweiligen Prozessregeln zurückgeführt werden. Die RPVM bildet damit den Grundstein zur Erfüllung der Anforderungen A3 und A4, also einer Unterscheidung in verpflichtenden und empfohlene Handlungen und deren Erklärung dem Prozessteilnehmer gegenüber.

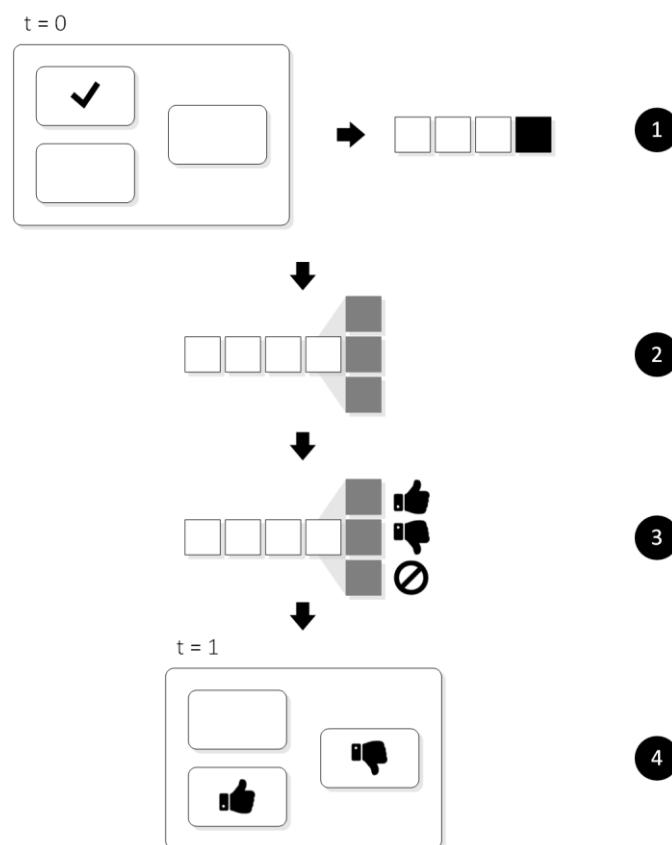


Abbildung 1.9 Ausführungsprinzip der *Rule-based Process Virtual Machine* (RPVM)

1.5.6 CMMN als regelbasierte Referenzsprache

Was BPMN für die prozedurale Welt ist, soll die *Case Management Model and Notation* (CMMN) für die regelbasierte Welt werden (Object Management Group 2014a). Der Standard wurde im Januar 2013 zum ersten Mal vorgeschlagen und befasst sich mit der regelbasierten Modellierung von Prozessen. Ein regelbasierter Prozess wird im Rahmen der CMMN als „Fall“ (engl. *case*) bezeichnet, weil der Standard ursprünglich aus dem Bereich der Fallbehandlung (engl. *case management*) stammt.

Obwohl der CMMN-Standard seit Mai 2014 in Version 1.0 vorliegt, weist er noch Schwächen bezüglich der oben genannten Anforderungen auf. CMMN unterstützt keine Modalitäten, so dass empfohlene nicht von verpflichtenden Regeln unterschieden werden können. Zudem lässt CMMN die Organisationsperspektive fasst vollständig außen vor und erlaubt auch nur bedingt eine perspektivenübergreifende Modellierung. Dennoch stellt CMMN den Stand der Entwicklung von regelbasierten Prozessmodellierungssprachen dar und soll daher als Referenzsprache zur grafischen, regelbasierten Prozessmodellierung dienen. Die Unterstützung von CMMN trägt zur Erfüllung von Anforderung A2, also der regelbasierten Modellierung agiler Prozesse, bei.

1.5.7 Deklarative, perspektivenübergreifende und multimodale Zwischensprache (DPIL) für regelbasierte Prozessmodellierungssprachen

Wie oben beschrieben kann CMMN die Anforderungen an eine regelbasierte Modellierungssprache nur bedingt erfüllen. Wie bereits für die prozedurale Ausführung soll daher auch die regelbasierte Komponente des PAS nicht für eine Sprache spezialisiert sondern von dieser entkoppelt werden. Um die Anforderungen an die regelbasierte Modellierung zu erfüllen, muss eine regelbasierte, perspektivenübergreifende und multimodale Zwischensprache entworfen werden. Diese Sprache wird im Folgenden *Declarative Process Intermediate Language* (DPIL) genannt. Sie deckt die fünf grundlegenden Perspektiven der Prozessbeschreibung ab. Die regelbasierte Komponente der Prozess-VM, die *Rule-based Process Virtual Machine* (RPVM) führt also Modelle in der Sprache DPIL aus. Die Sprache macht die Konzepte der RPVM für die Prozessmodellierung zugänglich und vervollständigt damit die Erfüllung der Anforderungen A2, A3, A5 und A6. Zur Unterstützung des Entwurfs von DPIL-Prozessen wird eine integrierte Entwicklungsumgebung für DPIL entworfen. Grafisch repräsentierte CMMN-Modelle können auf DPIL-Modelle abgebildet um sie auf der RPVM auszuführen.

1.5.8 Ausführung hybrider Prozessmodelle

Nach Anforderung A8 müssen sich strikte und agile Prozesse zu umfassenden Prozessmodellen kombinieren und integriert ausführen lassen. Eine Trennung auf der Basis von Schachtelungsebenen (Komposition) ist ein etablierter Weg um unterschiedliche Modellierungsprinzipien miteinander kombinieren zu können (Jablonski 1994; Sadiq et al. 2001; Pešić 2006). Hierbei kann für jeden zusammengesetzten Prozessschritt die Modellierungssprache unabhängig von der des umgebenden Prozesses gewählt werden. Bei dem in Abbildung 1.7 gezeigten Beispielprozess einer Klinik könnte so für die agilen Anteile eine regelbasierte Sprache verwendet werden, wodurch sich die Empfehlungen in das Modell integrieren ließen.

1.5.9 Entscheidungsunterstützung für Prozessteilnehmer

Wie oben beschrieben dienen vor allem agile Prozesse nicht mehr nur der Steuerung und Automatisierung sondern auch der Entscheidungsunterstützung menschlicher Prozessteilnehmer. Empfehlungen und Einschränkungen sollen für agile Prozesse in Form von Regeln formuliert werden. Dementsprechend können Verstöße gegen Empfehlungen dem Teilnehmer in Form von Verweisen auf die entsprechenden Regeln, also auf die relevanten Bereiche des Prozessmodells, zugänglich gemacht werden. Auf der Teilnehmeroberfläche werden empfohlene Handlungen also durch das zugrundeliegende Prozessmodell erklärt und damit Anforderung A4 erfüllt.

1.5.10 Zusammenfassung

In der vorliegenden Arbeit wird der Prototyp eines Ausführungssystems entwickelt, der die Ausführung prozeduraler und regelbasierter Modelle und deren Kombination unterstützt. Prozessmodellierungssprachen sind fortwährenden Änderungen unterworfen. Um das System von der auszuführenden Modellierungssprache zu entkoppeln, wird es als virtuelle Maschine konstruiert. Statt also die Ausführung einer konkreten Modellierungssprache zu implementieren, wird eine Plattform für eine Zwischensprache entwickelt. Um eine effiziente Ausführung sowohl prozeduraler als auch regelbasierter garantieren zu können, weist die Plattform jeweils

spezialisierte Anteile auf. Exemplarisch wird die Plattform für die Ausführung von BPMN- und CMMN-Prozessmodellen konfiguriert.

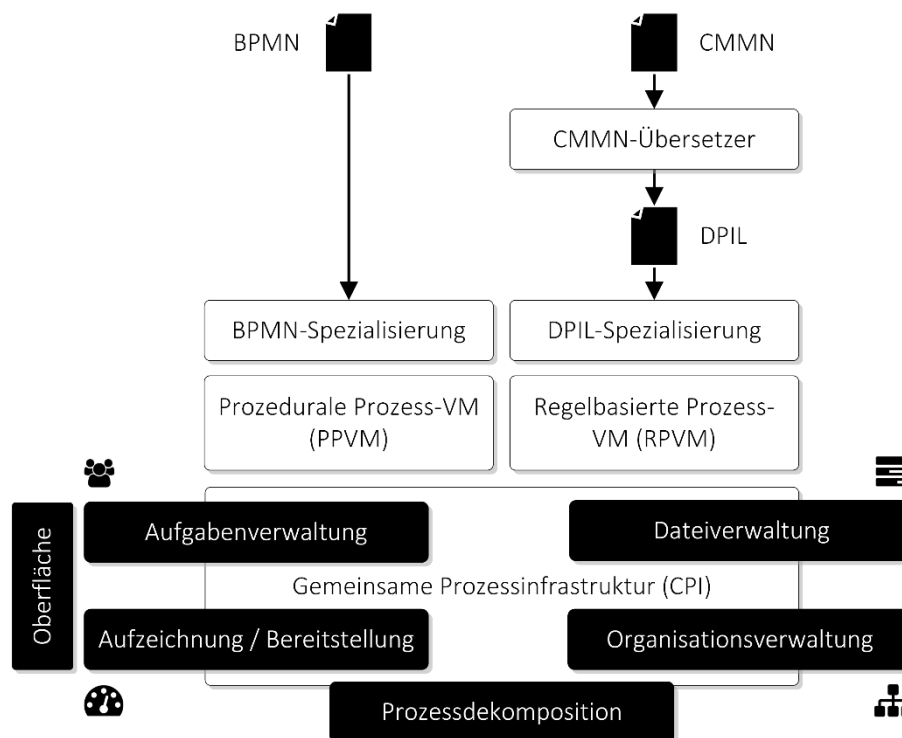


Abbildung 1.10 Angestrebte Gesamtarchitektur

Abbildung 1.10 zeigt die Gesamtarchitektur des entwickelten PAS. Ausgangspunkt bildet die gemeinsame Prozessinfrastruktur (CPI), die die Grundlage für die Ausführung von zusammengesetzten Prozessmodellen liefert. Sie integriert die Datei- und Organisationsverwaltung und beinhaltet die Aufgabenverwaltung und Schnittstellen zur Aufzeichnung und zur Bereitstellung von Prozessen.

Die Konzepte zur Ausführung prozeduraler Prozessmodelle sind bekannt und werden in der PPVM lediglich aufgegriffen und in leicht angepasster Form umgesetzt. Auch die konkrete Ausführung von BPMN-Modellen auf dieser Basis wird lediglich übernommen. Der individuelle Beitrag dieser Arbeit liegt im regelbasierten Anteil des Systems und in seiner Kombination mit dem prozeduralen. Die Idee ein Prozessausführungssystem als virtuelle Maschine zu realisieren wird beim Entwurf der RPVM aufgegriffen und auf eine regelbasierte Ausführung übertragen. Neu ist auch das Ausführungsprinzip der RPVM, die die möglichen Ereignisse eines Prozessmodells simuliert und je nach Bewertung dieser Ereignisse die Ausführung des Modells antreibt. Anders als bei bisherigen Ansätzen unterscheidet die RPVM dabei nicht zwischen Ereignissen für Aktivitäten und solchen etwa für Datenobjekte. Dies ermöglicht eine perspektivenübergreifende Modellierung von Prozessen im Sinne der Anforderungen. Folglich ist auch die Zwischensprache DPIL, die diese Konzepte der Modellierung zugänglich macht, ein innovativer Teil der Arbeit. Dies schließt die Abbildung der CMMN auf DPIL zur Ausführung ein. Die Konzepte der Sprache DPIL wurden erstmals in (Zeising et al. 2014) veröffentlicht. Die Funktionalitäten, die sowohl vom

prozeduralen, als auch vom regelbasierten Anteil des Systems genutzt werden, werden konsequent in einen weiteren Bestandteil (CPI) ausgelagert. Dabei entsteht eine Systemarchitektur, die eine tief integrierte Ausführung von prozeduralen und regelbasierten Modellen erlaubt, eine strikte Trennung von Zuständigkeiten umsetzt und in dieser Form ebenfalls innovativ ist. Das Prinzip der RPVM wurde erstmals in (Zeising et al. 2012) veröffentlicht.

2 Verwandte Arbeiten zur IT-gestützten Ausführung von Prozessmodellen

Im Folgenden wird ein Überblick über bestehende Ansätze zur IT-gestützten Ausführung von Prozessmodellen gegeben. Der deutlichste Unterschied zwischen den Ansätzen besteht im zugrundeliegenden Modellierungsprinzip. Deshalb wird grundsätzlich in prozedurale und regelbasierte Ansätze unterschieden. Anschließend werden Ansätze betrachtet, die beide Prinzipien kombinieren.

2.1 Prozedurale Modellierung

Die prozedurale Modellierung macht die möglichen Zustände, in denen sich der modellierte Prozess befinden kann und vor allem die Übergänge zwischen diesen Zuständen explizit. Zustände und Übergänge bilden also explizite Elemente des Modells und stehen damit im Vordergrund. Generell lassen sich dabei wiederum zwei Prinzipien unterscheiden: graphbasierte und strukturierte Sprachen. Graphbasierte prozedurale Modelle stellen Abläufe in einem gerichteten Graphen dar, wobei die Knoten meist einem Zustand und die Kanten einem zeitlichen Fluss entsprechen. Auch Verzweigungen und Zusammenführungen von Ablaufsträngen werden als Knoten modelliert. Strukturierte Sprachen hingegen stellen Abläufe mit Hilfe verschiedener Blöcke für Sequenzen, Verzweigungen und Schleifen dar. Im Folgenden werden die relevanten Vertreter der graphbasierten Modellierung und der einzige Vertreter der strukturierten Modellierung, WS-BPEL, untersucht.

2.1.1 Flussdiagramme

Die erste einheitlich festgelegte graphbasierte Sprache und außerdem Vorlage für nahezu alle heute verbreiteten Prozessmodellierungssprachen war das Flussdiagramm (European Computer Manufacturers Association (ECMA) 1966). Die Bausteine eines Flussdiagramms sind unter anderem der Prozessschritt (Rechteck), der Kontrollfluss (Pfeil), die Verzweigung (Raute) und die Anfangs- und Endpunkte (abgerundetes Rechteck). Der Prozessschritt symbolisiert eine Operation oder Aktion, der Kontrollfluss verbindet zwei zeitlich aufeinanderfolgende Operationen, an einer Verzweigung wird nur genau einer von mindestens zwei ausgehenden Flüssen gewählt und an den Anfangs- beziehungsweise Endpunkten beginnt beziehungsweise endet der Fluss und damit auch der Prozess. Ein exemplarisches Prozessmodell aus diesen Bausteinen ist in Abbildung 2.1 gezeigt. Das Modell sagt aus, dass nach der Bonitätsprüfung entweder die Ware ausgeliefert oder die Bestellung storniert wird.

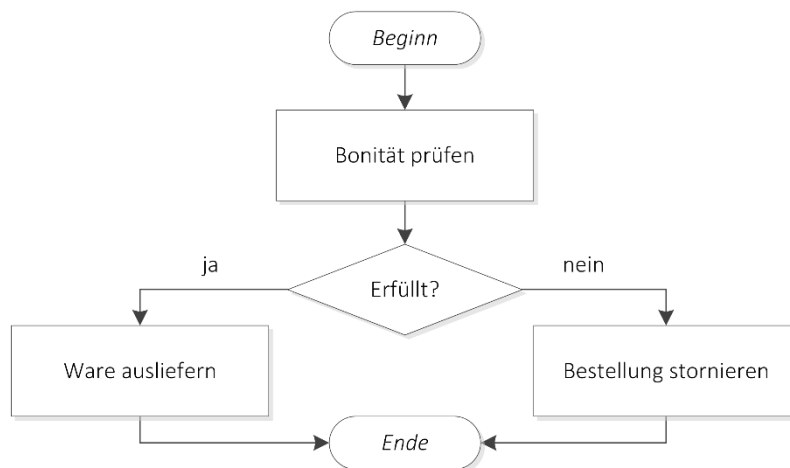


Abbildung 2.1 Exemplarisches Prozessmodell als Flussdiagramm

Im Folgenden werden die EPKs und die BPMN als die verbreitetsten Varianten des Flussdiagramms erläutert. Darüber hinaus existiert eine Vielzahl weiterer Ansätze, die aber zum einen weit weniger gebräuchlich sind und sich zum anderen von den hier gezeigten Ansätzen nicht wesentlich unterscheiden.

2.1.2 Ereignisgesteuerte Prozessketten (EPK)

Die *Ereignisgesteuerten Prozessketten* (EPK) entstanden 1992 in einem Forschungsprojekt mit der SAP AG an der Universität des Saarlandes in Saarbrücken. Ursprünglich deckte die Notation nur die funktionale, verhaltensbezogene und informationsbezogene Perspektive ab und wurde später als *Erweiterte EPK* (eEPK) um den organisatorischen und weitere Aspekte ergänzt.

Die zentralen Elemente einer EPK werden in Abbildung 2.2 gezeigt. Nachdem eine Bestellung erhalten wurde, wird die Bestellung überprüft und parallel dazu der Artikel produziert. An der Überprüfung ist die Bestellung als Datum beteiligt. Die Produktion des Artikels hat eine hohe Qualität zum Ziel und die Produktion als Organisationseinheit ist dafür zuständig. Nachdem die Bestellung bearbeitet und der Artikel produziert wurde, wird der Artikel ausgeliefert. Das Modell besteht damit aus Ereignissen ❶, Funktionen ❷, Operatoren ❸, Daten ❹, Organisationseinheiten ❺ und Zielen ❻.

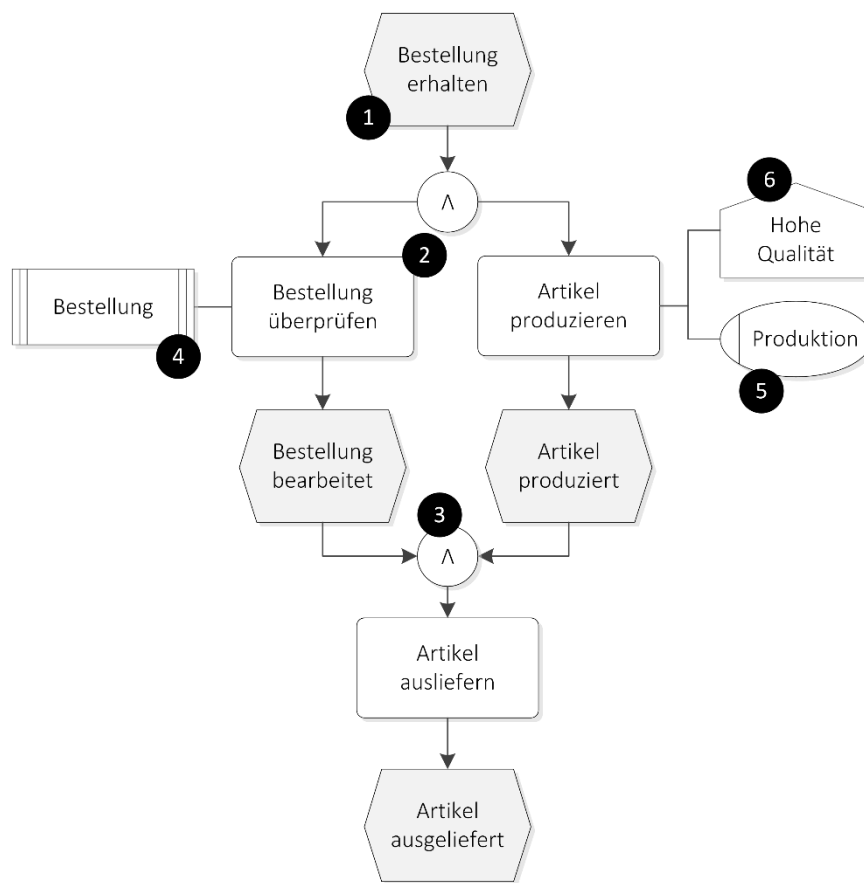


Abbildung 2.2 Exemplarisches EPK-Modell

EPKs werden ausdrücklich als abstrakte, „fachliche Modelle“ gesehen, die zunächst in „technische Modelle“ übersetzt werden müssen, bevor sie durch IT-Systeme ausgeführt werden können (Hoheisel et al. 2009). Durch diesen Übersetzungsschritt können EPKs nur bedingt als Kommunikationsmittel zwischen Fachabteilung und IT dienen. Dazu kommt, dass EPKs nicht formalisiert sind; es existiert also keine eindeutige Spezifikation (Decker et al. 2009). Damit können sie nur begrenzt maschinell verarbeitet werden.

Es ist davon auszugehen, dass der international etablierte BPMN-Standard die vor allem im deutschen Raum verbreiteten EPKs ablösen wird (Decker et al. 2009). Dies zeigt sich auch dadurch, dass die ursprüngliche Referenzimplementierung der EPKs, die ARIS-Plattform, selbst mittlerweile auch BPMN unterstützt (Software AG 2012). Außerdem ist die IDS Scheer Consulting GmbH, das Software- und Beratungshaus hinter der ARIS-Plattform, nun auch an der Standardisierung von BPMN beteiligt (Object Management Group Inc. 2011). EPKs treten also gegenüber BPMN in den Hintergrund und stellen nicht mehr den Stand der Technik in der prozeduralen Prozessmodellierung dar.

2.1.3 Business Process Model and Notation (BPMN)

Business Process Model and Notation (BPMN) ist ein Standard der Object Management Group (OMG) mit dem Ziel eine Notation für Prozessmodelle zu schaffen, die sowohl von den fachlichen Beteiligten als auch von den Software-Entwicklern verstanden werden kann.

BPMN besteht aus einem Metamodell und einer Visualisierung in Diagrammen. Die Visualisierung soll verständlich und das Metamodell vollständig sein, so dass sich Modelle maschinell interpretieren und direkt durch IT-Systeme unterstützen lassen. Diese Kombination aus reduzierter grafischer Darstellung und technisch detailliertem Modell bildet einen wichtigen Beitrag zur Vermittlung zwischen Geschäfts- und IT-Abteilungen (engl. *business / IT alignment*). Die grafische Darstellung bildet das Kommunikationsmittel zwischen fachlichen Beteiligten, während das darunterliegende Modell alle nötige Details zur technischen Implementierung des Prozesses transportiert.

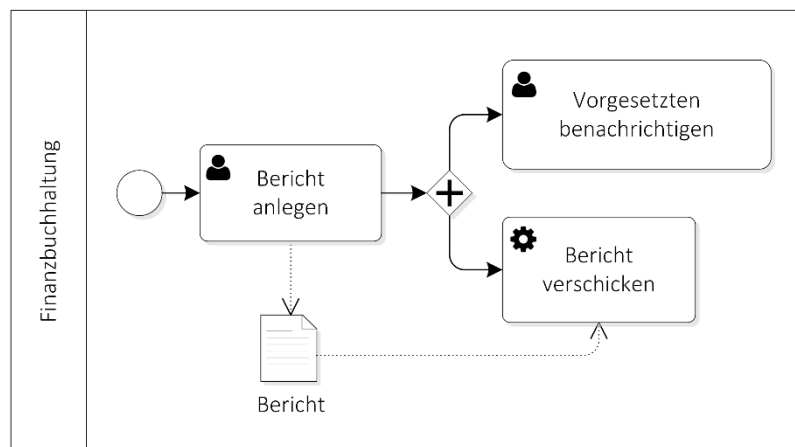


Abbildung 2.3 Exemplarisches BPMN-Diagramm

Als Beispiel für ein technisch detailliertes Modell wird im Folgenden das in XML serialisierte Modell zum Prozessdiagramm in Abbildung 2.3 aufgeführt. Ausgangspunkt bildet das **definitions-**Objekt als äußerstes Objekt und Einheit für den Austausch von BPMN-Modellen. Es enthält alle globalen Objekte wie den Prozess (**process**), die Elemente zur Anbindung des benötigten Webservices (**message**, **interface**, **itemDefinition** und **Operation**) und die Kollaboration (**Collaboration**), die in diesem Fall nur aus einem Teilnehmer (**participant**) und seinem Prozess besteht. Der Prozess enthält wiederum das Startereignis (**startEvent**), die personelle Aufgabe (**userTask**), den Aufruf des Webservices (**serviceTask**), das parallele Gatter (**parallelGateway**), das Datenobjekt (**dataObject**) und die Sequenzflüsse (**sequenceFlow**). Die Datenobjekte sind über Datenverbindungen (**dataInputAssociation**, **dataOutputAssociation**) mit den Aufgaben verbunden. Die im folgenden Quelltext grau hinterlegten Elemente haben keine Entsprechung im Diagramm, stellen also technische Details dar. Die Details zum Beispiel zum Aufruf des Webservices (**message**, **operation** etc.) sind zwar im Quelltext aber nicht im Diagramm enthalten. Im Anschluss an den Prozess enthält das Modell eine Serialisierung des oben gezeigten Diagramms (**BPMNDiagram**). Hier werden die Formen mit ihren Maßen und Koordinaten aufgelistet und mit den Modellelementen verknüpft, die sie darstellen (nicht gezeigt). Das oben dargestellte Diagramm ist also eine grafische Zusammenfassung des folgenden Quelltexts.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
  xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  expressionLanguage="http://www.w3.org/1999/XPath"
  id="_1394189174212"

targetNamespace="http://sourceforge.net/bpmn/definitions/_1394189174212"
  typeLanguage="http://www.w3.org/2001/XMLSchema">
  <itemDefinition id="ID_1" isCollection="false" itemKind="Information"
    structureRef="xsd:string"/>
  <message id="MSG_1" itemRef="ID_1" name="Report"/>
  <interface id="IF_1" name="ReportService"
    implementationRef="acme:publishService">
    <operation id="IF_1_O_1" name="PublishReport"
      implementationRef="acme:doPublish">
      <inMessageRef>MSG_1</inMessageRef>
    </operation>
  </interface>
  <collaboration id="COLLABORATION_1">
    <participant id="_2" name="Finanzbuchhaltung" processRef="PROCESS_1"/>
  </collaboration>
  <process id="PROCESS_1" isExecutable="true">
    <startEvent id="_3">
      <outgoing>_9</outgoing>
    </startEvent>
    <userTask id="_4" name="Bericht anlegen">
      <incoming>_9</incoming>
      <outgoing>_10</outgoing>
      <ioSpecification>
        <dataOutput id="Dout_4_15" isCollection="false"/>
        <inputSet/>
        <outputSet>
          <dataOutputRefs>Dout_4_15</dataOutputRefs>
        </outputSet>
      </ioSpecification>
      <dataOutputAssociation id="_16">
        <sourceRef>Dout_4_15</sourceRef>
        <targetRef>_15</targetRef>
      </dataOutputAssociation>
    </userTask>
    <parallelGateway gatewayDirection="Diverging" id="_5">
      <incoming>_10</incoming>
      <outgoing>_11</outgoing>
      <outgoing>_12</outgoing>
    </parallelGateway>
    <userTask id="_6" name="Vorgesetzten benachrichtigen">
      <incoming>_11</incoming>

```

```

    </userTask>
    <serviceTask id="_7" name="Bericht verschicken"
operationRef="IF_1_0_1">
    <incoming>_12</incoming>
    <ioSpecification>
    <dataInput id="Din_7_15" isCollection="false"
itemSubjectRef="ID_1"/>
    <inputSet>
    <dataInputRefs>Din_7_15</dataInputRefs>
    </inputSet>
    <outputSet/>
    </ioSpecification>
    <dataInputAssociation id="_17">
    <sourceRef>_15</sourceRef>
    <targetRef>Din_7_15</targetRef>
    </dataInputAssociation>
    </serviceTask>
    <sequenceFlow id="_9" sourceRef="_3" targetRef="_4"/>
    <sequenceFlow id="_10" sourceRef="_4" targetRef="_5"/>
    <sequenceFlow id="_11" sourceRef="_5" targetRef="_6"/>
    <sequenceFlow id="_12" sourceRef="_5" targetRef="_7"/>
    <dataObject id="DO_PROCESS_1_1" isCollection="false" name="Bericht"/>
    <dataObjectReference dataObjectRef="DO_PROCESS_1_1" id="_15"/>
  </process>
  <bpmndi:BPMNDiagram> ... </bpmndi:BPMNDiagram>
</definitions>

```

BPMN deckt die funktionale Perspektive durch zusammengesetzte Prozesse (Subprozesse) und elementare Aktivitäten (Aufgaben) ab. Die Verhaltensperspektive wird durch den Sequenzfluss, Gatter und Ereignisse abgedeckt. Die Informationsperspektive wird unter anderem durch Datenobjekte und Datenverbindungen abgedeckt. Zur Abdeckung der Organisationsperspektive kann für eine personelle Aufgabe ein Ausdruck hinterlegt werden, der den oder die jeweils zuständigen Benutzer liefert. Im Zusammenhang mit der Operationsperspektive werden unterschiedliche Realisierungen für elementare Aktivitäten, also die verschiedenen Aufgabentypen angeboten. Dazu zählen die personelle Aufgabe für menschliche Prozessteilnehmer, der Aufruf eines Dienstes, der Start eines Skripts, die Auswertung von Geschäftsregeln und andere. Damit werden alle grundlegenden Perspektiven abgedeckt. Ein ausführbares BPMN-Modell muss über einen konsistenten Sequenzfluss verfügen. Zur Ausführung wird prinzipiell zunächst der Sequenzfluss ausgewertet und alle weiteren Aspekte sind nachgelagert. Alle Abhängigkeiten wie zum Beispiel Datenflüsse oder organisatorische Abhängigkeiten müssen sich also explizit im Sequenzfluss wiederfinden.

BPMN kann als führender Standard zur prozeduralen Prozessmodellierung gelten und repräsentiert daher in dieser Arbeit den aktuellen technischen Stand dieses Modellierungsprinzips. Prozedurale Prozessmodelle sollen in dieser Arbeit mit Hilfe des BPMN-Standards notiert werden.

2.1.4 Petri-Netze

Um Prozesse analysieren zu können, müssen sie in einer wohldefinierten und eindeutigen Sprache formuliert werden. Eine solche Sprache bilden die Petri-Netze. Der Formalismus wurde in den 60er Jahren entwickelt und seit dem um zahlreiche Konzepte erweitert. Ein Prozess, der mittels Petri-Netzen modelliert wurde, besitzt eine klare und präzise Bedeutung, da die Semantik von Petri-Netzen und deren Erweiterungen mit formalen Mitteln festgelegt wurden. Das mathematische Fundament ermöglicht außerdem den Nachweis bestimmter Eigenschaften (Blockaden usw.) und die Anwendung verschiedener Metriken (Wartezeiten usw.) (van der Aalst 1998).

Abbildung 2.4 zeigt einen einfachen BPMN-Prozess und seine Entsprechung als Petri-Netz. Durch die beiden parallelen Gatter (Rauten mit Plus-Symbol) in ① werden nach der Aufgabe A gleichzeitig B und C aktiviert und vor D findet eine Synchronisation statt. Die Aufgaben B und C werden also nebenläufig durchgeführt. Dasselbe Verhalten lässt sich mit dem Petri-Netz in ② erreichen. Die Aufgaben wurden in Transitionen (Quadrate) und die Gatter und Ereignisse in entsprechende Stellen (Kreise) überführt (van der Aalst 1998). Die Marke (ausgefüllter Kreis) stellt den aktuellen Zustand des Petri-Netzes dar.

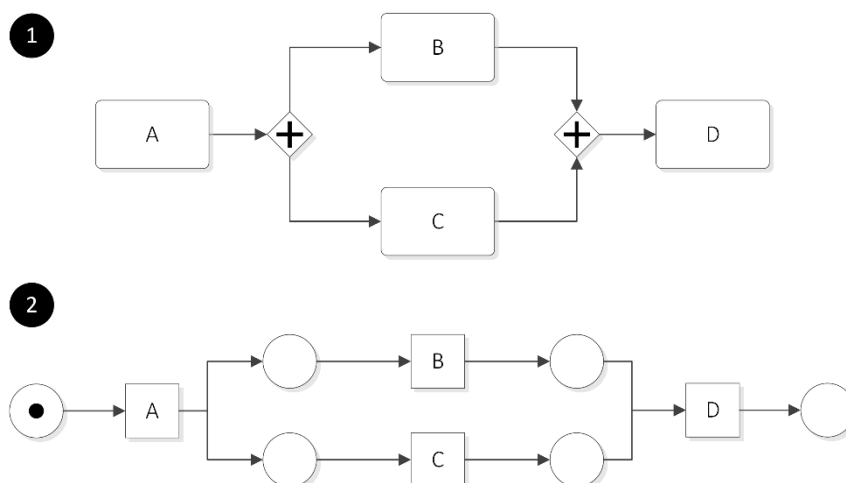


Abbildung 2.4 Einfaches BPMN-Modell und seine Entsprechung als Petri-Netz

Petri-Netze können dazu dienen, Elementen der Verhaltensperspektive zu definieren oder Aspekte dieser Perspektive zu analysieren. Die Elemente der Informations-, Organisations-, Funktions- und Operationsperspektive können in Petri-Netzen zum großen Teil nicht abgebildet werden. Sie werden daher im Rahmen dieser Arbeit nicht als Prozessmodellierungssprache behandelt. Die Idee, prozedurale Modelle mittels der Bewegungen von Marken zu interpretieren, geht aber auf eben diesen Formalismus zurück. Auch im Rahmen dieser Arbeit sollen prozedurale Modelle mittels dieses Prinzips interpretiert und ausgeführt werden.

2.1.5 BPEL als strukturierte Prozessmodellierungssprache

Die *WS-Business Process Execution Language* (*WS-BPEL* oder kurz *BPEL*) ist eine Sprache, mit der sich das Zusammenspiel von Diensten in Form von Prozessen spezifizieren lässt (OASIS 2007a). Da einer der Dienste den Prozess kontrolliert, spricht man von einer „Orchestrierung“ von Diensten

(Josuttis 2008). Die Sprache ging 2002 aus Microsoft XLANG und IBM WSFL hervor und reiht sich in die Familie der „WS-*“-Standards ein. Diese Standards stellen allesamt Erweiterungen für Dienste dar, die auf der Kombination aus dem *Simple Object Access Protocol* (SOAP) und *Web Services Description Language* (WSDL) basieren. Bis zur Verbreitung des *Representational State Transfer* (REST)-Protokolls stellten Dienste auf SOAP- und WSDL-Basis die bedeutendste technische Umsetzung einer dienstorientierten Architektur (engl. *service-oriented architecture*, kurz SOA) dar (Weerawarana et al. 2005).

Bei BPEL handelt es sich um eine strukturierte Sprache; Abläufe werden also aus verschachtelten Sequenz-, Verzweigungs- und Schleifenblöcken gebildet. Die Erweiterungen *WS-BPEL4People* (OASIS 2007b) und *WS-HumanTask* (OASIS 2010) ermöglichen die Beschreibung von personellen Aufgaben und damit die Einbindung der Organisationsperspektive. Um wiederverwendbare Subprozesse modellieren, also die funktionale Perspektive vollständig abdecken zu können, wurde die Erweiterung *BPEL-SPE* (Kloppmann et al. 2005) vorgeschlagen. Die Erweiterung wurde allerdings bisher nicht technisch spezifiziert und kann daher auch nicht durch Werkzeuge unterstützt werden.

Im folgenden Beispiel wird eine Sequenz aus drei Schritten beschrieben. Die *receive*-Aktivität wartet auf eine eingehende Nachricht, die *invoke*-Aktivität ruft eine Operation auf und die über BPEL4People eingebundene *task*-Aktivität aus der WS-HumanTask-Spezifikation realisiert eine personelle Aufgabe.

```
<process name="purchaseOrderProcess" ...>
  ...
  <sequence>
    <receive partnerLink="purchasing" operation="sendPurchaseOrder"
      variable="order" createInstance="yes">
      <documentation>Receive Purchase Order</documentation>
    </receive>
    <invoke partnerLink="shipping" operation="requestShipping"
      inputVariable="shippingRequest" outputVariable="shippingInfo">
      <documentation>Decide On Shipper</documentation>
    </invoke>
    <extensionActivity>
      <b4p:peopleActivity name="prepareShippingDocuments">
        <ht:task/>
      </b4p:peopleActivity>
    </extensionActivity>
  </sequence>
</process>
```

Es existieren zwar vereinzelt herstellerspezifische grafische Notationen für BPEL-Prozesse, eine standardisierte Notation ist allerdings nicht vorgesehen. Das mag vor allem daran liegen, dass die von XML vorgegebene Blockstruktur für eine übersichtliche Darstellungsweise ungeeignet ist. BPEL ist also eher als technische Programmiersprache zu verstehen und richtet sich vornehmlich an Software-Entwickler. BPEL ist daher als prozedurale Modellierungssprache für Endbenutzer

und für die Vermittlung zwischen Fachabteilung und IT ungeeignet. Die meisten Organisationen, die bereits an der Spezifikation von BPEL beteiligt waren, sind nun auch an der von BPMN beteiligt (OASIS 2007a) (Object Management Group Inc. 2011). Es ist also zu erwarten, dass BPEL als bedeutendste strukturierte Sprache zur prozeduralen Prozessmodellierung von BPMN verdrängt wird.

2.2 Regelbasierte Modellierung

Sobald die möglichen Zustände eines Prozesses und die Übergänge zwischen ihnen nicht mehr expliziter Teil des Prozessmodells sind, handelt es sich nicht mehr um ein prozedurales Modell (Fahland et al. 2009). Die direkten Vorgänger und Nachfolger eines bestimmten Zustands lassen sich dann nicht mehr wie zum Beispiel in einem Petri-Netz über direkte Transitionen erreichen sondern ergeben sich aus übergreifenden Randbedingungen. Es besteht kein expliziter Zusammenhang mehr zwischen dem Modell und seinem Verhalten. Die entsprechenden Ansätze werden unter der „regelbasierten“ Modellierung zusammengefasst.

Im Folgenden werden die bestehenden Ansätze zur regelbasierten Modellierung ausführbarer Prozessmodelle näher beleuchtet.

2.2.1 GPSG

Die *Generalised Process Structure Grammars* (GPSG) sind ein früher Ansatz zur regelbasierten Modellierung von Prozessen (Glance et al. 1996). Die Besonderheit an diesem Konzept ist, dass die Regeln eines GPSG-Prozesses sowohl die Bearbeitung von Aktivitäten als auch den Zugriff auf Dokumente gleichermaßen einschränken können. Die organisatorische Perspektive wird nicht behandelt, es wird nicht zwischen verpflichtenden und empfohlenen Regeln unterschieden und es bleibt unklar, inwieweit das Konzept implementiert werden kann. Die GPSG können also nicht direkt zur Unterstützung von agilen Prozessen eingesetzt werden. Dennoch soll die Idee einer Verbindung von Aktivitäten und Dokumenten in übergreifenden Prozessregeln im Rahmen dieser Arbeit aufgegriffen werden.

2.2.2 DECLARE

DECLARE ist ein Rahmenwerk zur Modellierung und Ausführung von regelbasierten Prozessmodellen. Innerhalb des Rahmenwerks lassen sich Regelschablonen definieren und zu einer Sprache zusammenfassen. Die beiden häufig genannten Sprachen sind *ConDec* und *DecSerFlow*. Um die Modelle ausführen zu können, müssen die Regeln auf einen maschineninterpretierbaren Formalismus abgebildet werden. Bisher wurden im Rahmen von DECLARE zwei Formalismen vorgeschlagen: die Lineare Temporale Logik (LTL) (Pešić 2006) und das Ereigniskalkül (engl. *event calculus*, kurz EC) nach Kowalski und Sergot (Montali et al. 2009). Für jede Regelschablone muss eine Entsprechung im jeweiligen Formalismus hinterlegt werden. Die Prozessmodelle können so in eine validierbare und ausführbare Form übersetzt werden.

Ein Beispiel ist die Regel „wenn a durchgeführt wird, muss irgendwann danach auch b durchgeführt werden“. Diese Regel kann als Funktion `response(a, b)` ausgedrückt werden. Als grafische Repräsentation dieser Funktion ist ein Pfeil mit einem Punkt am Anfang wie in Abbildung 2.5 für die konkreten Aktivitäten A und B vorgesehen.

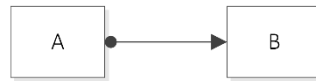


Abbildung 2.5 Grafische Darstellung der response-Regel in DECLARE

Die Abbildung der oben genannten Regel auf einen LTL-Ausdruck lautet $\Box(a \rightarrow \Diamond b)$. Mit Hilfe der LTL lassen sich Aussagen über Folgen von Zuständen formulieren. Das Symbol \Box sagt aus, dass der Ausdruck in Klammern für jeden Zustand der Folge gilt. Das Symbol \Diamond sagt aus, dass der folgende Ausdruck in irgendeinem Zustand der Folge gilt. Die Formel sagt also, dass das Eintreten von a immer irgendwann das Eintreten von b impliziert.

Die LTL-Formel muss zur Interpretation wiederum in einen Automaten wie in Abbildung 2.6 überführt werden, der dann durchlaufen werden kann und die erlaubten Zustandsübergänge enthält (Pešić 2006).

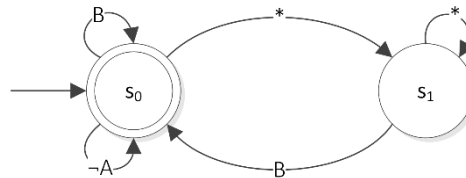


Abbildung 2.6 Automat zum DECLARE-Beispiel

Die Regeln in DECLARE setzen Aktivitäten nur zeitlich in Beziehung zueinander ohne andere Perspektiven einzubeziehen. Diese Regeln können im Ganzen zwar wiederum von beliebigen Bedingungen abhängen. Dadurch lassen sich organisations- oder informationsbezogene Aspekte aber nur zum Teil darstellen. Die Regel **response(A, B)** mit der Bedingung „ $x < 3$ “ fordert, dass wenn A durchgeführt wurde, auch irgendwann danach B durchgeführt werden muss, falls der Wert der Variable x kleiner drei ist. Regeln können andere Perspektiven außer dem Kontrollfluss also nicht beeinflussen. Ein Zusammenhang wie „*wenn A von einem Auszubildenden durchgeführt wurde und der Bericht noch nicht vorliegt, dann muss irgendwann danach B von seinem Vorgesetzten durchgeführt werden*“ lässt sich damit nicht direkt ausdrücken. DECLARE unterstützt somit keine perspektivenübergreifende Modellierung im Sinne der Anforderungen. Die Idee, den Verlauf eines Prozesses durch logische Regeln einzuschränken, soll jedoch im Rahmen dieser Arbeit wieder aufgegriffen werden.

2.2.3 ESProNa

Die *Engine for Semantic Process Navigation* (ESProNa) wurde von 2009 bis 2011 am Lehrstuhl für Angewandte Informatik IV der Universität Bayreuth entwickelt (Igler 2011) (Igler, Jablonski, et al. 2010) (Igler, Faerber, et al. 2010). Es handelt sich um ein Expertensystem für regelbasierte Prozessmodelle. Das System ist in der Logiksprache Prolog implementiert und Prozessmodelle müssen ebenfalls in Prolog verfasst werden. Das System unterstützt zwei Arten von Anfragen. Zum einen lässt sich prüfen, ob ein gegebener Zustand den Regeln des Prozessmodells entspricht, was sich zur Ausführung von Prozessmodellen nutzen lässt. Zum anderen lässt sich der günstigste Pfad von einem Ausgangs- zu einem Zielzustand ermitteln. Dieser Teil des Systems wird als Navigation bezeichnet.

Die Navigation ist aufgrund der kombinatorischen Explosion allerdings ohne weiteres nicht praktikabel. Das verwendete heuristische Suchverfahren erfordert eine feingranulare Gewichtung der Regeln, also umfangreiche Informationen und Annahmen zum Prozess. Die Regelsprache von ESProNa ist nicht definiert oder eingegrenzt. Es handelt sich vielmehr um eine Programmbibliothek, mit der sich regelbasierte Prozesse beschreiben lassen. Der Sprachumfang ist der von Prolog, wodurch sich Prozessmodelle nicht validieren sondern lediglich auf korrekte Prolog-Syntax prüfen lassen. ESProNa stellt selbst auch keine Workflow-Engine dar (Workflow Management Coalition 1999), sondern deckt lediglich die Regelauswertung ab. Es fehlt zum Beispiel die Verwaltung von Prozessinstanzen, Regeln für die Terminierung von Instanzen und die Schachtelung von Prozessmodellen. ESProNa kann also nicht direkt zur IT-Unterstützung agiler Prozesse eingesetzt werden. Die Idee, den Zustand einer Prozessinstanz als Vektor von vergangenen Ereignissen darzustellen, soll jedoch im Rahmen dieser Arbeit wieder aufgegriffen werden.

2.2.4 EM-BrA²CE

Geschäftsregeln (engl. *business rules*) sollen das Verhalten und die Informationen einer Organisation beeinflussen oder führen (zur Muehlen et al. 2008). Sie spezifizieren nur die Randbedingungen, die eingehalten werden sollen und nicht, wie genau ein Ziel erreicht werden kann. Geschäftsregeln und Prozesse sind derzeit also voneinander unabhängige Konzepte. Die *Semantics of Business Vocabulary and Business Rules* (SBVR) ist ein durch die OMG standardisiertes Rahmenwerk für Geschäftsregeln (Object Management Group 2013). EM-BrA²CE macht den ersten Schritt zur Vereinigung von Geschäftsregeln und Prozessen. Es erweitert die SBVR um Konzepte wie Aktivitäten, Zustände und Bearbeiter und ermöglicht es damit, Prozesse in SBVR auszudrücken (Goedertier & Vanthienen 2007). Die Schwierigkeit liegt allerdings in der tatsächlichen Durchsetzung (*enforcement*) dieser Regeln, also im Fall von EM-BrA²CE in der Ausführung der Prozesse. Zur Ausführung müssen die SBVR-Regeln mittels Schablonen in ECA-Regeln (*event condition action*) überführt werden. So wird die SBVR-Schablone

`<Activity2> may only happen after <Activity1>`

in die ECA-Regel

`on start(<Activity2>): if not completed(<Activity1>) then report violation`

übersetzt (De Roover et al. 2012). Da nur die in einer Schablone erfassten Geschäftsregeln übersetzt und ausgeführt werden können, geht der ursprüngliche Vorteil durch den Umfang von SBVR verloren. Die Prozesse werden also effektiv mit ECA-Regeln modelliert. Außerdem wird nicht gezeigt, wie mit verschiedenen Modalitäten in den SBVR-Regeln bei deren Übersetzung umgegangen wird. EM-BrA²CE kann also nicht direkt zur IT-Unterstützung von agilen Prozessen verwendet werden. Die Idee, Geschäftsregeln und Prozesse zu regelbasierten Prozessmodellen zusammenzuführen, wird jedoch im Rahmen dieser Arbeit aufgegriffen.

2.2.5 CMMN

Die *Case Management Model and Notation* (CMMN) wurde von der OMG Anfang 2013 das erste Mal vorgeschlagen. Die Motivation ist, dass ein Fall (engl. *case*) anders als ein Prozess nicht als

vorgegebene Sequenz von Aktivitäten abgearbeitet werden kann, weil diese Sequenz nicht ad hoc bekannt ist, sondern erst durch Erfahrung etabliert und außerdem von Fall zu Fall sehr verschieden sein kann. (Object Management Group 2014a) Ein Fall im Sinne der CMMN entspricht also dem, was oben als agiler Prozess beschrieben wurde. Um Fälle durch IT unterstützen zu können, definiert die CMMN eine regelbasierte Sprache. Die Struktur eines Falls (Case) wird im Wesentlichen mittels Aufgaben (Task), den Elementen der Fallakte (CaseFileItem) und Meilensteinen (Milestone) modelliert. Die Teilnehmer werden über Rollen (Role) mit den personellen Aufgaben (HumanTask) des Falls verknüpft.

Regeln werden durch Wachen (Sentry) realisiert, bei denen es sich um Tripel aus Ereignis, Bedingung und Aktion (engl. *event condition action*, kurz ECA) handelt. Tritt das Ereignis ein und ist die Bedingung erfüllt, dann wird die Aktion ausgelöst. Ereignisse können sich dabei auf Zustandsübergänge von Elementen der Fallakte (CaseFileItemTransition) oder von Aufgaben, Meilensteinen oder Ereignissen (PlanItemTransition) beziehen. Die Bedingung einer Regel kann sich lediglich auf die Fallakte beziehen. Die resultierende Aktion ist das Betreten oder Verlassen einer Aufgabe, eines Abschnitts oder eines Meilensteins.

CMMN beinhaltet also die üblichen Perspektiven der Prozessmodellierung (vgl. Abschnitt 1.3.5). Prozesse lassen sich durch Regeln einschränken. Dabei können aber als Auslöser nur Ereignisse der Verhaltens- und Informationsperspektive, als Bedingung nur Zusammenhänge der Informationsperspektive und als Konsequenz nur die Verhaltensperspektive eingesetzt werden. CMMN erlaubt daher nur bedingt eine perspektivenübergreifende Modellierung. Prozesse lassen sich also nicht basierend auf der Organisationsperspektive und nicht hinsichtlich der Informations- und Organisationsperspektive einschränken.

Zudem lassen sich Teilnehmer nur über fest vorgegebene Rollen zur Entwurfszeit den Aufgaben zuweisen. Im Diagramm wird dies nicht dargestellt. Die Organisationsperspektive ist in CMMN somit nur ansatzweise abgedeckt. Zudem unterstützt CMMN keine Modalitäten. Regeln lassen sich nicht hinsichtlich ihrer Qualität klassifizieren und werden somit stets als verbindlich interpretiert. Empfehlungen lassen sich in CMMN nicht transportieren.

Trotz dieser Einschränkungen kann CMMN als Stand der Technik gesehen werden, was die regelbasierte Prozessmodellierung angeht. Wie auch BPMN kombiniert CMMN eine reduzierte grafische Darstellung in Diagrammen mit einem technisch detaillierten Modell und ist daher gut für die Vermittlung zwischen Fach und IT geeignet. Im Rahmen dieser Arbeit gilt CMMN als fortschrittlichster Standard zur grafischen regelbasierten Prozessmodellierung. Um dem Rechnung zu tragen, wird das entwickelte PAS auch in der Lage sein, CMMN-Modelle auszuführen.

2.3 Hybride Modellierungskonzepte

Ziel dieser Arbeit ist eine umfassende Unterstützung von strikten und agilen Prozessen durch IT und damit eine Ausführung von Kombinationen aus prozeduralen und regelbasierten Prozessmodellen. Im Folgenden werden die Ansätze beleuchtet, die dieses Ziel ebenfalls verfolgen.

2.3.1 MOBILE

Im Zusammenhang mit dem MOBILE-Projekt wurde bereits 1994 eine Mischung aus prozeduralen und regelbasierten Modellen vorgestellt. Prozedurale MOBILE-Prozesse enthalten die gewohnten Konstrukte für serielle, alternative und parallele Ausführung. Regelbasierte Prozesse können die Regeln Frist (*deadline*), Verzögerung (*delay*) und Existenz (*existence*) enthalten. Mit diesen Regeln lassen sich Aktivitäten zeitlich ordnen und ihr Auftreten einschränken. Beide Arten von Prozessen können ineinander verschachtelt werden (Jablonski 1994). Die regelbasierten Prozesse werden allerdings nur skizziert und in darauf folgenden Arbeiten nicht mehr erwähnt. Zudem setzen die drei Regeln die Aktivitäten nur zeitlich in Beziehung ohne andere Perspektiven einzubeziehen. Eine perspektivenübergreifende regelbasierte Modellierung erlaubt das Konzept somit nicht. Das MOBILE-System kann daher nicht direkt zur IT-Unterstützung von strikten und agilen Prozessen eingesetzt werden. Die Kombination von prozeduralen und regelbasierten Modellen mittels einer Schachtelung von Prozessen wird jedoch im Rahmen dieser Arbeit wieder aufgegriffen.

2.3.2 Pockets of Flexibility

Das Konzept der „Pockets of Flexibility“ erlaubt in prozeduralen Prozessen spezielle Teilprozesse, in denen kein Kontrollfluss definiert wird und die Ausführungsreihenfolge der Aktivitäten beliebig ist. Das Konzept erlaubt innerhalb dieser Teilprozesse allerdings keinerlei Regeln und eine umgekehrte Schachtelung ist nicht vorgesehen (Sadiq et al. 2001). Dennoch wird auch hier wieder die Notwendigkeit der Kombination strikter und agiler Prozesse anerkannt. Zudem wird die Kombination der beiden Prinzipien über die Schachtelung von Modellen aufgegriffen.

2.3.3 DECLARE & YAWL

YAWL ist eine prozedurale Prozessmodellierungssprache mit entsprechendem Ausführungssystem. Das Ziel von YAWL ist eine möglichst große Abdeckung von prozeduralen Mustern, also zum Beispiel komplexen Schleifen und Synchronisationsmechanismen (van der Aalst et al. 2003). YAWL sieht die Integration beliebiger Dienste vor. Zum einen kann YAWL Aktivitäten an einen Dienst delegieren. Zum anderen kann ein Dienst wiederum den Start von Prozessinstanzen in YAWL anstoßen. Durch die Integration von YAWL und DECLARE nach diesem Prinzip lassen sich Kombinationen aus YAWL- und DECLARE-Prozessen ausführen (Pešić 2006). Die Kombination von prozeduralen und regelbasierten Modellen wird auch hier wieder durch die Schachtelung von Prozessmodellen erreicht. Wie oben erwähnt eignet sich DECLARE jedoch bereits nicht zur Unterstützung agiler Prozesse im Sinne der Anforderungen. Somit ist auch die Kombination aus YAWL und DECLARE nicht geeignet um sie zu erfüllen.

2.3.4 Geschäftsregeln in prozeduralen Prozessen

Geschäftsregeln (engl. *business rules*) sollen das Verhalten und die Informationen einer Organisation beeinflussen oder führen (zur Muehlen et al. 2008). Sie lassen dabei die verhaltensbezogene Perspektive naturgemäß außen vor (Goedertier & Vanthienen 2007), spezifizieren also nur die Randbedingungen, die eingehalten werden sollen und nicht wie genau ein Ziel erreicht werden kann. Geschäftsregeln und Prozesse sind in der Praxis derzeit strikt getrennte Welten, die nur an bestimmten Punkten miteinander verknüpft sind. Der übliche Weg besteht in der Auslagerung komplexer Entscheidungen in Geschäftsregeln (Stiehl 2013).

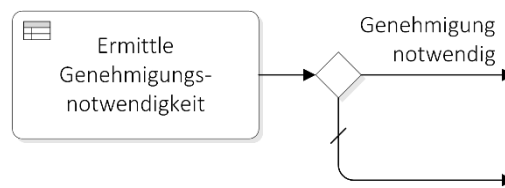


Abbildung 2.7 Integration von Geschäftsregeln in einen BPMN-Prozess (vgl. (Stiehl 2013))

BPMN sieht hierfür die Geschäftsregelaufgabe (*business rule task*) vor. Inhaltlich gesehen kapselt sie eine komplexe Entscheidung. Technisch beschreibt sie den Aufruf eines *Business Rule Management Systems* (BRMS). Das Ergebnis der Entscheidung liefert die Grundlage für eine nachfolgende Verzweigung (Object Management Group Inc. 2011). Ein Beispiel hierfür ist in Abbildung 2.7 gezeigt. Die komplexe Entscheidung über die Notwendigkeit einer Genehmigung wird nicht im Prozess selbst modelliert sondern in eine Geschäftsregel ausgelagert. Vom Regelsystem wird dann zum Beispiel anhand der in Tabelle 2.1 gezeigten Entscheidungstabelle entschieden. Der Ausgang der Entscheidung steuert dann den weiteren Verlauf des Prozesses.

Tabelle 2.1 Entscheidungstabelle zum Beispiel für Geschäftsregeln (vgl. (Stiehl 2013))

Land	Bestellwert	Genehmigung notwendig?
EMEA	< 100.000	false
	≥ 100.000	true
APJ	< 70.000	false
	≥ 70.000	true
AMER	< 150.000	false
	≥ 150.000	true

Das bedeutendste Rahmenwerk zur Abbildung von Geschäftsregeln ist die *Semantics of Business Vocabulary and Business Rules* (SBVR) (Object Management Group 2013). SBVR erlaubt die Spezifikation von Zusammenhängen und Regeln in einer an Englisch angelehnten natürlichen Sprache. Eine Alternative dazu bietet die Im Februar 2014 vorgestellte *Decision Model and Notation* (DMN) (Object Management Group 2014b). Mit DMN lassen sich Entscheidungen mittels Diagrammen modellieren.

Geschäftsregeln bilden derzeit also keine Prozesse ab, sondern kapseln lediglich gewisse logische Bestandteile prozeduraler Prozesse. Obwohl hier also das prozedurale (strikte Prozessmodelle) und das regelbasierte Prinzip (Geschäftsregeln) kombiniert werden, kann die Kombination nicht zur Unterstützung strikter und agiler Prozesse dienen, weil Geschäftsregeln allein nicht zur Modellierung von Verhalten dienen können.

2.4 Zusammenfassung

Aus dem Flussdiagramm haben sich die vor allem im deutschen Raum verbreiteten EPKs und die international gebräuchliche BPMN entwickelt. Die EPKs werden aufgrund ihrer Schwächen zunehmend von der BPMN verdrängt. Die BPMN ist aufgrund ihres prozeduralen Charakters besonders zur Modellierung von strikten Prozessen geeignet und gilt in diesem Bereich als Stand

der Technik. Sie deckt alle grundlegenden Perspektiven einer Prozessbeschreibung ab und kombiniert eine grafische Darstellung mit einem detaillierten Metamodell. Die BPMN bildet daher im Rahmen dieser Arbeit die Referenz zur prozeduralen Modellierung von strikten Prozessen. Petri-Netze gelten als wichtigstes Werkzeug zur formalen Interpretation prozeduraler Modelle, werden im Rahmen dieser Arbeit aber nicht als eigenständige Prozessmodellierungssprache eingestuft.

Im Bereich der regelbasierten Prozessmodellierung kann DECLARE als bedeutendster Beitrag zur Modellierung und Ausführung gelten. Die Regeln eines DECLARE-Modells entsprechen jedoch letztlich nur einem regelbasierten Kontrollfluss. Eine Verschränkung der grundlegenden Modellierungsperspektiven erlaubt DECLARE nur bedingt und kann daher nicht als Ansatz zur Unterstützung von agilen Prozessen übernommen werden.

Analog zu BPMN wurde außerdem die CMMN zur regelbasierten Prozessmodellierung vorgeschlagen. Die CMMN deckt die grundlegenden Perspektiven zumindest ansatzweise ab und kombiniert wie die BPMN eine grafische Darstellung mit einem technisch detaillierten Metamodell. Die CMMN kann im Rahmen dieser Arbeit als Stand der Technik im Bereich der regelbasierten Prozessmodellierung gelten. Da die CMMN aufgrund ihrer Schwächen jedoch zu viele der Anforderungen nicht erfüllen kann, soll sich die vorliegende Arbeit im Bereich der agilen Prozesse nicht allein auf die CMMN beschränken.

Die bestehenden Lösungen, die sowohl prozedurale als auch regelbasierte Prozessmodelle abdecken, entsprechen nicht den Anforderungen und können die Prozesslandschaft einer Organisation daher nur eingeschränkt unterstützen.

Ein PAS, das das gesamte Spektrum einer Organisation von strikten zu agilen Prozessen unterstützt, muss also in der Lage sein, Kombinationen zumindest aus BPMN- und CMMN-Modellen auszuführen. Da weder BPMN noch CMMN die oben genannten Anforderungen vollständig abdeckt, soll das PAS jedoch nicht an die beiden spezifischen Sprachen gebunden werden. Vielmehr soll es eine Zwischenschicht bereitstellen, die die Anforderungen erfüllt und auf die die genannten Sprachen zur Ausführung abgebildet werden.

3 Gemeinsame Prozessinfrastruktur (CPI) der Process Navigation Plattform

Im Folgenden werden sowohl das konzeptionelle Fundament als auch eine prototypische Implementierung einer Plattform zur integrierten IT-gestützten Ausführung von strikten und agilen Prozessen beschrieben. Das PAS wird *Process Navigation* (PN) genannt und soll die oben erhobenen Anforderungen erfüllen und die Schwächen der bestehenden Ansätze überwinden.

Aufgrund der unterschiedlichen Ausführungsprinzipien prozeduraler und regelbasierter Systeme benötigt die PN-Plattform zwei entsprechend spezialisierte Kerne, die prozedurale und die regelbasierte Prozess-VM (siehe Abschnitte 4 und 5). Ein großer Teil ihrer Aufgaben ist aber unabhängig vom Modellierungsprinzip. Diesen gemeinsam genutzten Teil der Funktionalität – also die Grundlage für die Ausführung von Prozessmodellen unterschiedlicher Modellierungssprachen – bildet die *Common Process Infrastructure* (CPI). Sie integriert außerdem Dienste wie die Aufgaben-, Datei- und Organisationsverwaltung. Diese Dienste gehören nicht zum Kern des PAS und stellen austauschbare Komponenten mit definierten Schnittstellen dar (vgl. Abbildung 3.1). Sowohl die Kernfunktionalitäten als auch die Schnittstellen werden im Folgenden genauer betrachtet.

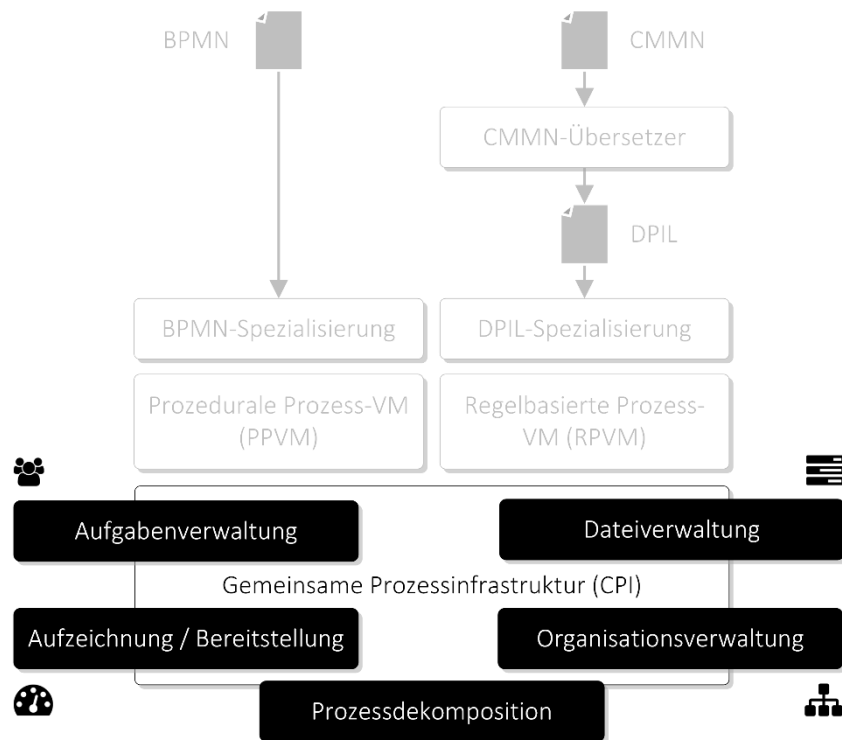


Abbildung 3.1 Aufbau der PN-Plattform mit CPI

3.1 Vorbemerkung zu objektorientierten Entwurfsmustern

Der Entwurf objektorientierter Software muss spezifisch für das jeweilige Problem und dennoch allgemein genug sein, um zukünftigen Problemen und Anforderungen begegnen zu können. Erfahrene Entwickler lösen ein Problem nicht von Grund auf neu, sondern nutzen Lösungen, die auch in der Vergangenheit schon funktioniert haben. Demnach findet man in ihren Systemen wiederkehrende Muster von Klassen und Objekten, die bestimmte Entwurfsprobleme lösen und das System letztlich flexibel, elegant und wiederverwendbar machen. Ein Entwickler, der diese Muster kennt, kann sie unmittelbar anwenden, ohne sie erst entdecken zu müssen. Die objektorientierten Entwurfsmuster fassen häufige und bewährte Muster objektorientierter Systeme zusammen (Gamma et al. 1994). Auch beim Entwurf der PN-Plattform treten viele Anforderungen auf, die durch bewährte Entwurfsmuster gelöst werden können.

3.2 Grundsätzliches Ausführungsprinzip

Die *Process Virtual Machine* (PVM) nach Baeyens und Faura stellt eine Basisarchitektur zur Ausführung von Prozessmodellen dar (Baeyens & Faura 2007). Sie ist zwar ausdrücklich auf die Ausführung graphbasierter prozeduraler Modelle wie BPMN ausgerichtet, dient aber im Rahmen dieser Arbeit dennoch als Vorlage für das grundsätzliche Ausführungsprinzip jeglicher Prozessmodelle.

Wie bereits erwähnt, kann die Ausführung prozeduraler Modelle als Bewegung von Marken durch einen Prozessgraphen implementiert werden. Die PVM greift dieses Prinzip auf und übersetzt ein Prozessmodell zunächst in einen Graphen aus Knoten (**Node**) und gerichteten Kanten (**Transition**), durch den sich ein Ausführungszeiger (**Execution**) bewegt und damit den aktuellen Ausführungszustand darstellt. Knoten entsprechen vor allem Aktivitäten aber auch Gattern und Ereignissen. Die Kanten entsprechen dem Kontrollfluss. Jeder Knoten verfügt über ein bestimmtes Verhalten (**Executable**), das benachrichtigt wird, sobald der Knoten durch den Zeiger betreten oder verlassen wird. Sowohl Knoten als auch Zeiger können verschachtelt werden (**nestedNodes** und **children**), wodurch die Architektur auch verschachtelte Prozessmodelle unterstützt. Sowohl die Knoten des Prozessmodells als auch die Ausführungszeiger sind dabei als *Kompositum* beschrieben. Sowohl die einzelnen Objekte als auch Kompositionen aus ihnen können also einheitlich behandelt werden (Gamma et al. 1994).

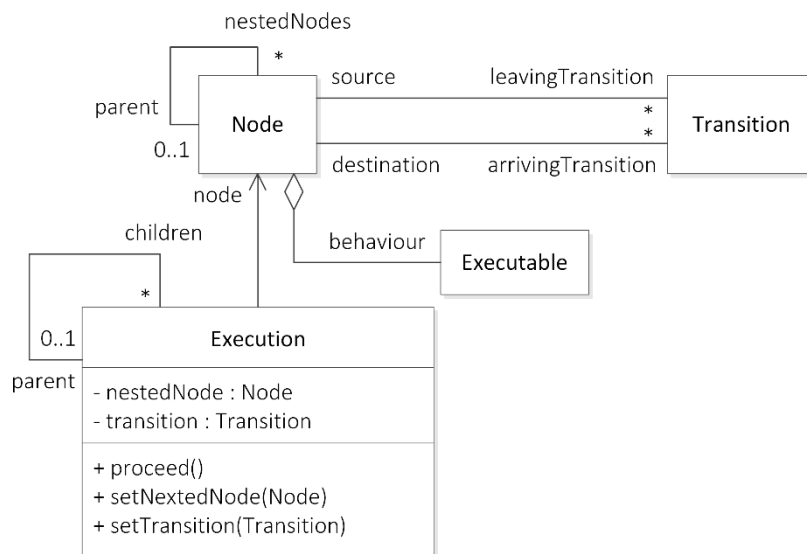


Abbildung 3.2 Kern des Process Virtual Machine-Musters nach Baeyens und Faura (vgl. (Baeyens & Faura 2007))

Die PVM soll als Ausgangspunkt für die CPI-Ebene des PN dienen. Dabei müssen wesentliche Aspekte verändert werden, so dass sie als Basisfunktionalität zur Ausführung prozeduraler und regelbasierter Prozessmodelle dienen kann. Die resultierende Architektur ist in Abbildung 3.3 dargestellt. Die wesentlichen Unterschiede zur PVM nach Baeyens und Faura werden im Folgenden beschrieben.

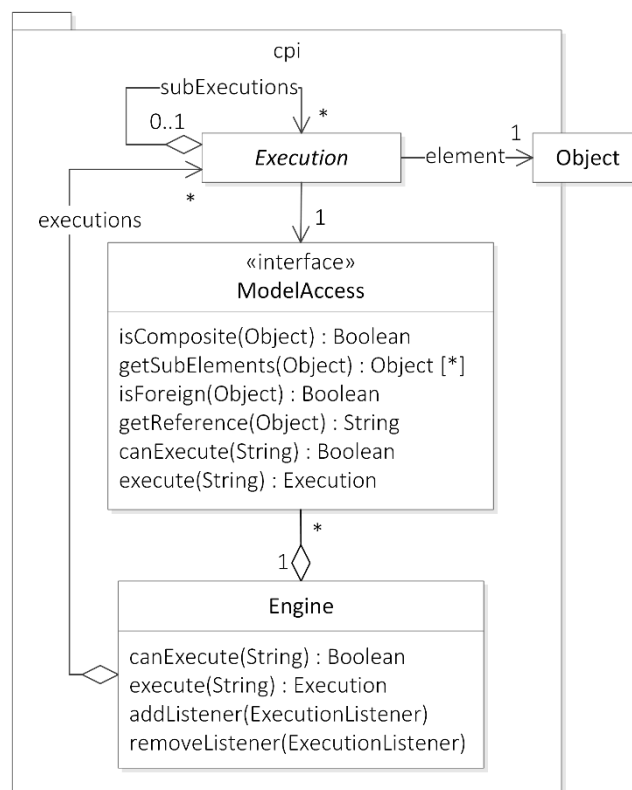


Abbildung 3.3 Ausführungszeiger der CPI für zusammengesetzte Prozessmodelle

3.2.1 Elemente statt Knoten und Kanten

Aktivitäten, also die Knoten eines Prozessgraphen der PVM, existieren sowohl in prozeduralen als auch in regelbasierten Modellen. Kontrollflüsse, also die Kanten der PVM, sind allerdings nur in prozeduralen Modellen sinnvoll und können daher nicht Teil der CPI als Basis beider Prinzipien sein. Die Abstraktion besteht darin, dass ein Ausführungszeiger der CPI lediglich Modellelemente ohne besondere Spezifika durchläuft statt Knoten und Kanten. Elemente, die spezifisch für prozedurale Modelle sind, entfallen dadurch und die CPI ist im Gegensatz zur PVM nicht an ein bestimmtes Modellierungsprinzip gebunden.

3.2.2 Direkte Ausführung des Prozessmodells

Für Notationen wie BPMN liegt das Metamodell bereits in maschinenlesbarer Form als XML-Schema-Dokument vor. Der Vorteil hiervon wird in Abbildung 3.4 gezeigt. Aus dem Schema (1) lassen sich wiederum Klassen generieren (2). BPMN-Prozessmodelle liegen als XML-Dokumente vor, die dem BPMN-Schema entsprechen (3). Durch Deserialisierung lässt sich ein solches Modell in Objekte der generierten Klassen überführen (4). Der entstehende Objektgraph repräsentiert das Prozessmodell und kann programmatisch unmittelbar verwendet, also zum Beispiel von einem Ausführungszeiger (5) referenziert werden.

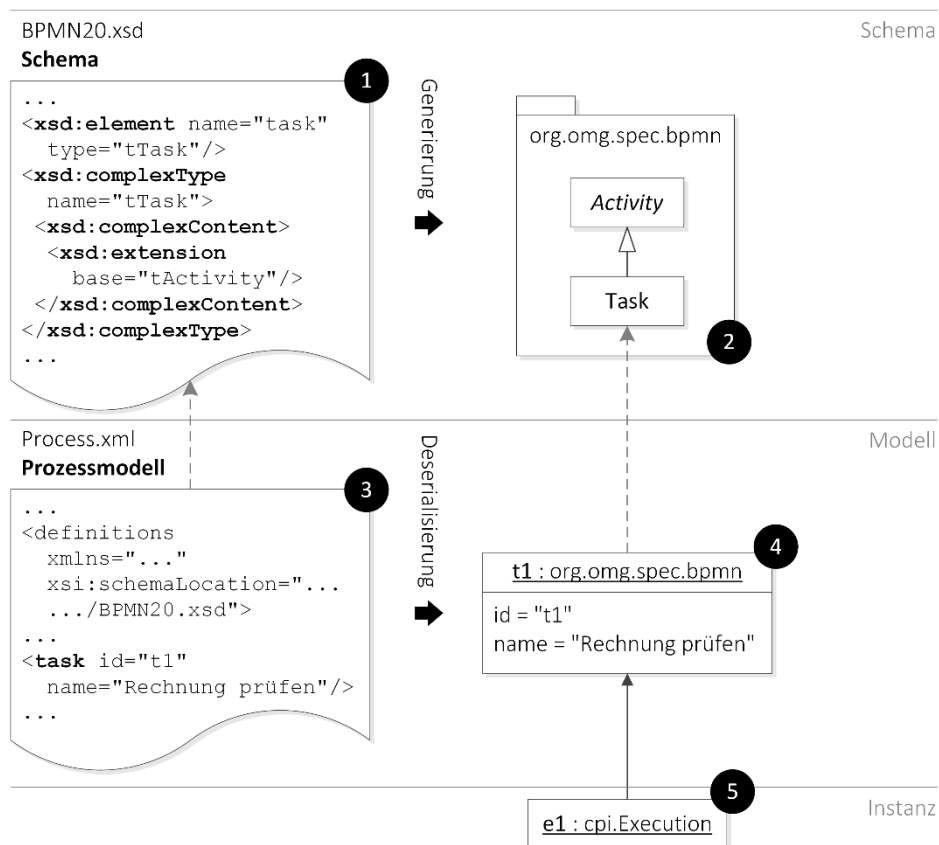


Abbildung 3.4 Direkte Ausführung von BPMN-Modellen durch Generierung von Klassen aus dem XML-Schema und Deserialisierung von Modellobjekten.

Die PVM erfordert vor der Ausführung eines Prozessmodells (Abbildung 3.4 ④ angedeutet) zunächst dessen Transformation in einen Graphen aus **Node**- und **Transition**-Objekten (siehe Abbildung 3.2). Ein solcher Graph stellt lediglich eine Abstraktion des ursprünglichen Modells dar und unterscheidet sich strukturell nicht vom ursprünglichen Modell. Die Transformation stellt also einen unnötigen Aufwand dar. Die Ausführungszeiger der CPI sollen die entsprechenden Modellobjekte daher direkt durchlaufen können. Die entsprechende Typisierung des Verweises auf das aktuelle Element kann dadurch entfallen (**e**lement in Abbildung 3.3 im Gegensatz zu **n**ode in Abbildung 3.2). Der Ausführungszeiger, der sich direkt durch das Prozessmodell bewegt, realisiert das Entwurfsmuster eines *Interpreters*. Er wertet das Modell aus, indem er es nach einem definierten Prinzip durchwandert (Gamma et al. 1994).

3.2.3 Ausführung kombinierter Prozessmodelle

Zusammengesetzte Prozessschritte dienen zur Strukturierung von umfangreichen Prozessmodellen und zur Wiederverwendung von Teilprozessen. In den meisten Modellierungssprachen handelt es sich um einen Prozessschritt, der auf einen untergeordneten Prozess verweist. Dieser Mechanismus wird auch genutzt, um unterschiedliche Modellierungsprinzipien zu kombinieren (Jablonski 1994) (Sadiq et al. 2001) (Pešić 2006).

Die Architektur der PVM unterstützt bereits zusammengesetzte Prozessschritte. Erreicht ein Ausführungszeiger einen zusammengesetzten Knoten, so werden weitere Ausführungszeiger für dessen untergeordnete Knoten erzeugt und damit das untergeordnete Modell ausgeführt. In einem Prozessgraphen der PVM kann ein zusammengesetzter Knoten direkt auf die untergeordneten Knoten verweisen (Verknüpfung **nestedNodes** in Abbildung 3.2). Um die Kombination verschiedener Modellierungssprachen zu unterstützen, muss dieser Verweis verallgemeinert werden. Jeder Ausführungszeiger der CPI wird hierzu mit einem Objekt für den sprachspezifischen Zugriff auf Modellelemente (**ModelAccess**) konfiguriert. Mit ihm lassen sich Verschachtelungen auflösen (**isComposite** und **getSubElements**) und Verweise auf Modelle in anderen Sprachen behandeln (**isForeign**, **getReference**, **canExecute** und **execute**). Die Auslagerung dieses Verhaltens entspricht dem Entwurfsmuster einer *Strategie*, da die verschiedenen **ModelAccess**-Instanzen eine Familie austauschbarer Algorithmen bilden (Gamma et al. 1994). Für jede Sprache muss eines dieser Objekte instanziiert und beim Vermittler (**Engine**) registriert werden. Durch diese Architektur bleibt es vor der CPI verborgen, wie ein zusammengesetzter Prozessschritt den untergeordneten Prozess referenziert. Die CPI ist dadurch in der Lage Kombinationen beliebiger Prozessmodellierungssprachen auszuführen.

Ein Beispiel soll dazu dienen, die Ausführung von kombinierten Prozessmodellen zu illustrieren. In BPMN kann eine Aufrufaktivität (engl. *call activity*) modelliert werden (Object Management Group Inc. 2011), die zum Beispiel auf einen regelbasierten Prozess in der Sprache DPIL verweist. Die CPI hat keine Kenntnis über die Semantik einer Aufrufaktivität und muss daher über die Eigenschaft **isForeign** (Abbildung 3.3) feststellen, ob das Element einen Verweis auf ein fremdes Modell darstellt. Mit **getReference** kann über die BPMN-spezifische Implementierung des Modellzugriffs (**BpmnAccess**) das Ziel des Verweises ermittelt werden. Die tatsächliche Realisierung von Verweisen bleibt damit vor der CPI verborgen. Für jede auf dem System auszuführende Sprache muss bei einem Vermittler (**ModelAccessBroker**) ein entsprechendes

Zugriffsobjekt hinterlegt werden. Das Verweisziel wird an den Vermittler übergeben, der das Zugriffsobjekt auswählt, das den Verweis interpretieren und das Modell ausführen kann. Das DPIL-spezifische Zugriffsobjekt (`DpilAccess`) erzeugt einen neuen regelbasierten Ausführungszeiger für den DPIL-Prozess. Dieser gelangt zurück zum Ausführungszeiger des BPMN-Prozess und wird diesem untergeordnet.

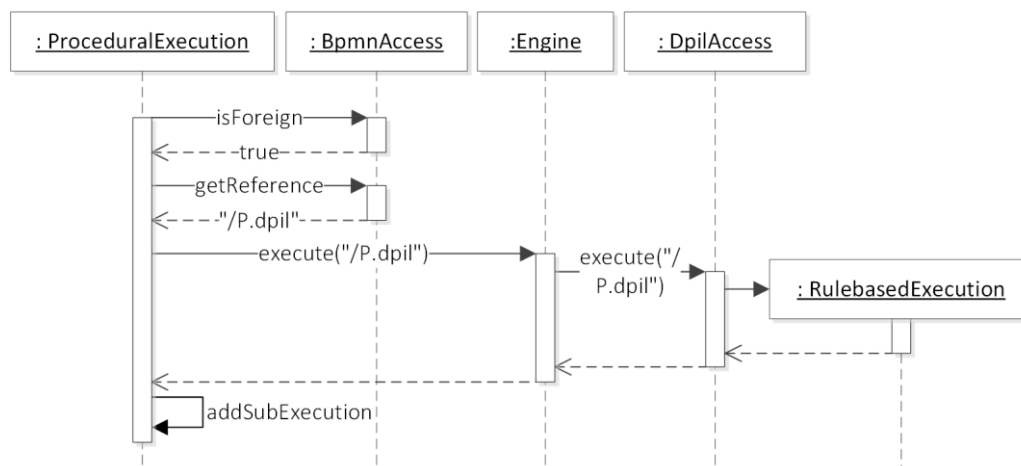


Abbildung 3.5 Ablauf der Auflösung eines verschachtelten Prozessmodells

3.3 Bewertungen und Regeln

Wie bereits erwähnt soll die PN-Plattform besonders bei der Ausführung agiler Prozesse zur Entscheidungsunterstützung dienen und dazu empfohlene oder nicht empfohlene Aktionen auf der Basis des Prozessmodells erklären. Eine solche Erklärung wird im Folgenden „Bewertung“ (engl. *rating*) genannt. Da die Prozessteilnehmer auf der Ebene der CPI mit der PN-Plattform kommunizieren, muss sie in der Lage sein, Bewertungen zu transportieren. Bewertungen wären auch im Kontext prozeduraler Modelle, also strikter Prozesse, denkbar. Durch die Verankerung in der CPI würde die PN-Plattform sie für prozedurale Modelle auch in gleicher Form unterstützen. Dennoch wirft die Bewertung einzelner Konstrukte in einer Sprache wie BPMN eine Vielzahl an Fragen auf, die über den Rahmen dieser Arbeit hinausgehen. Daher werden an dieser Stelle Bewertungen nur im Zusammenhang mit regelbasierten Prozessmodellen betrachtet.

Abbildung 3.6 zeigt, dass eine Bewertung (`Rating`) als eine Menge von verletzten Regeln (`Rule`) definiert ist. Die innere Struktur einer Regel (`Rule`) bleibt hier verborgen. Lediglich die Art der Regel (harte Regel, weiche Regel oder Meilenstein) und ein Hinweistext bilden die entsprechende Schnittstelle. Eine Aktion wird als nicht erlaubt bewertet (`isAllowed`), wenn eine oder mehrere harte Regeln verletzt werden. Die Änderung der Bewertung (`RatingDifference`) zum Beispiel einer Aktion ist der Unterschied von einer Bewertung (`before`) zu einer anderen (`after`). Dementsprechend sind verletzte Regeln hinzugekommen (`addedViolatedRules`) und weggefallen (`removedViolatedRules`). Eine Aktion kann „nicht erlaubt“, „neutral“, „empfohlen“ oder „nicht empfohlen“ sein, je nachdem welche Arten von Regeln hinzugekommen oder weggefallen sind.

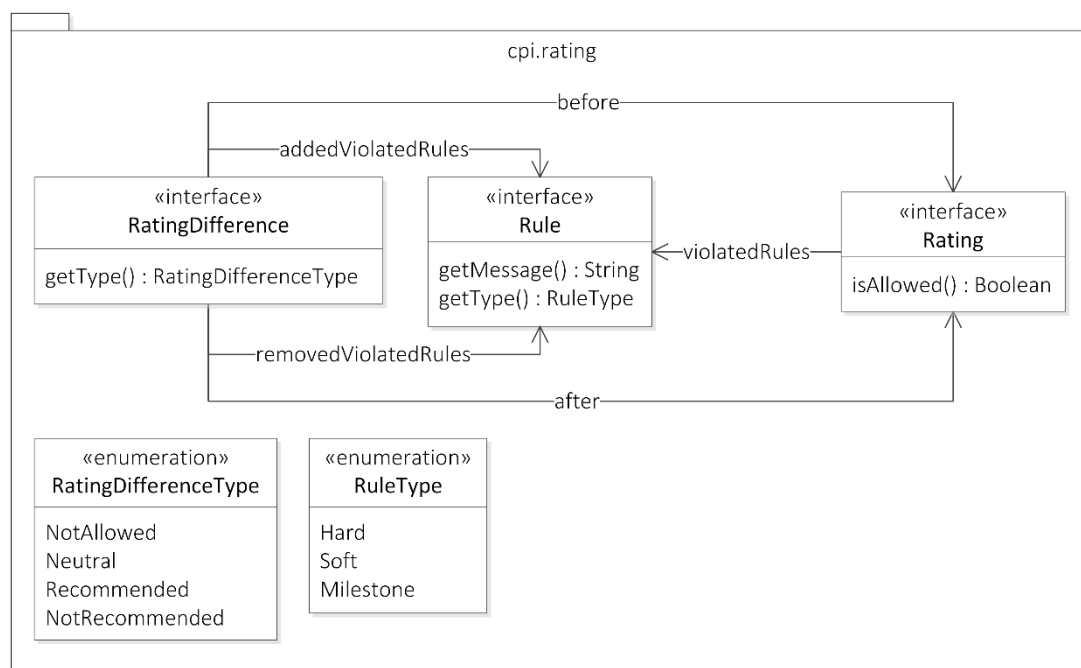


Abbildung 3.6 Schnittstelle der CPI zu Regeln und Bewertungen

Die in Abbildung 3.6 gezeigten Elemente bilden die Schnittstelle der PN-Plattform bezüglich Bewertungen. Die Implementierung dieser Schnittstelle wird im Rahmen des regelbasierten Teils der PN-Plattform in Abschnitt 5 beschrieben.

3.4 Dateiverwaltung

Gemäß dem Prinzip der Trennung von Zuständigkeiten (Dijkstra 1982) verwaltet ein PAS dauerhafte prozessrelevante Inhalte nicht selbst sondern integriert zu diesem Zweck ein *Enterprise Content Managment* (ECM)-System. Ein solches System deckt den Lebenszyklus von Inhalten wie zum Beispiel Dokumenten und Bildern innerhalb einer Organisation ab. Zu den Kernkomponenten des ECM zählen u.a. (Gilbert et al. 2013):

Digitalisierung: Diese Komponente deckt die Erfassung, Verarbeitung und Verwaltung von digitalisierten Papierdokumenten ab. Hierzu zählen Technologien für das Scanning, zur Texterkennung und Verarbeitung von Formularen.

Dokumentenverwaltung: Die Kernfunktionalitäten umfassen eine Revisionsverwaltung, Sperrmechanismen, eine Zugriffskontrolle und die Indizierung von Dokumenten. Weiterführende Funktionalitäten sind die Verwaltung zusammengesetzter Dokumente und die Replikation von Inhalten.

Archivierung: Durch Automatisierungstechniken und Richtlinien wird eine langfristige Aufbewahrung von Inhalten ermöglicht. Dadurch lassen sich entsprechende rechtliche, regulatorische und betriebliche Vorgaben erfüllen. Angestrebt wird vor allem die Übereinstimmung mit Standards wie der *Department of Defense Directive* 5015.2-STD, *The National Archives* (TNA), der *Victorian Electronic Records Strategy* (VERS) und den *Model Requirements for the Management of Electronic Records* (MoReq2).

Web Content Management (WCM): Durch den Einsatz von Web-Technologien für den Zugriff auf das Kernrepositorium eines ECM-Systems werden zusätzliche Funktionalitäten möglich. Dazu zählen Werkzeuge zur Erzeugung von Inhalten, der Einsatz von Vorlagen und die Bereitstellung von Inhalten über einen Webserver.

Soziale Medien: Dieser Aspekt umfasst das Teilen von Dokumenten, Zusammenarbeit, Wissensverwaltung und die Unterstützung von projektbezogenen Arbeitsgruppen. Wichtige Komponenten sind Blogs, Wikis und die Unterstützung für bedeutende Online-Plattformen.

Geschäftsprozesse: Geschäftsprozesse sollen unterstützt werden, indem Inhalte verteilt, Aufgaben zugewiesen und Protokolle aufgezeichnet werden. Im einfachsten Fall umfasst dieser Aspekt die Prüfung und Genehmigung von Dokumenten. Dass die Autoren die Unterstützung von Prozessen zu den Kernkomponenten eines ECM-Systems zählen, zeigt, dass ein ECM-System erst in Kombination mit einem Prozessmanagementsystem seinen vollen Nutzen entfaltet.

Der *Content Management Interoperability Services* (CMIS)-Standard ist eine herstellerübergreifende und sprachunabhängige Spezifikation zur Kommunikation mit ECM-Systemen (OASIS 2012). Mit Hilfe von CMIS lassen sich Lösungen konstruieren, die mit mehreren ECM-Systemen arbeiten. Außerdem werden ECM-Kunden weniger abhängig vom jeweiligen Hersteller und der Migrationsaufwand sinkt.

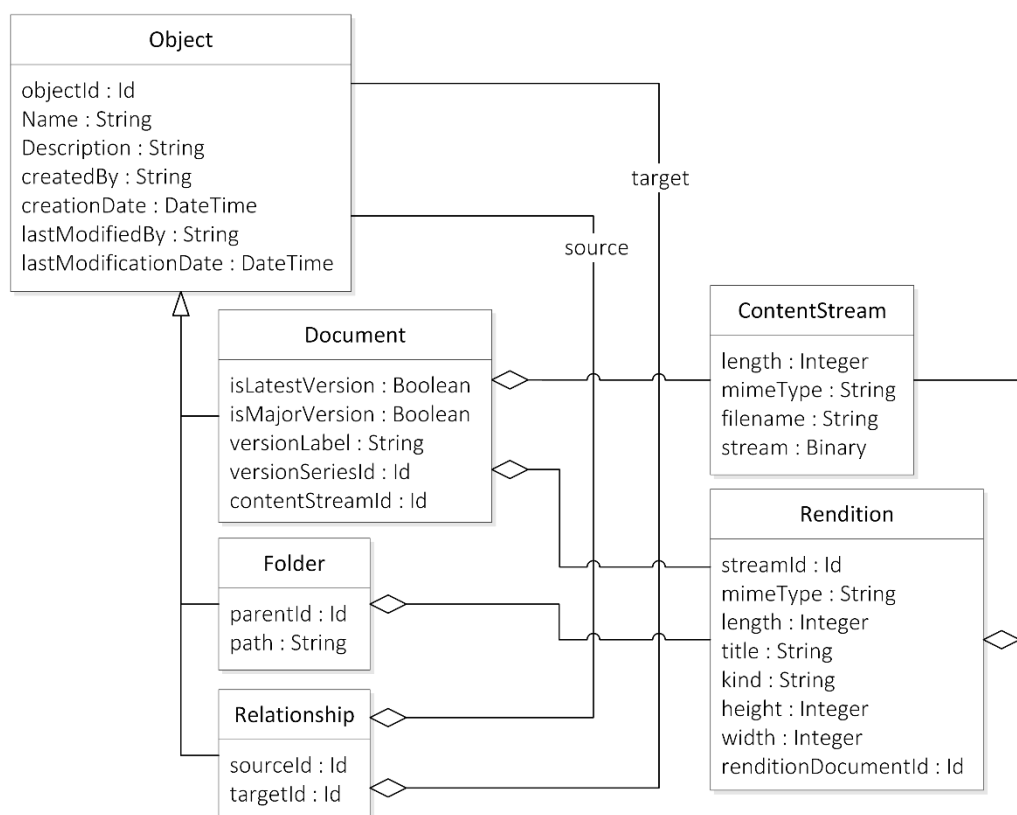


Abbildung 3.7 Kernelemente des CMIS-Datenmodells (vgl. (OASIS 2012))

Der Standard wird mittlerweile von allen namhaften Systemen implementiert und es wird erwartet, dass die breite Akzeptanz des Standards auf die ECM-Industrie ähnliche Auswirkungen hat wie die SQL-Standardisierung auf die Datenbank-Industrie (Müller et al. 2013).

Das logische Modell, das von einem CMIS-kompatiblen ECM-System verwaltet wird, ist in Abbildung 3.7 auszugsweise gezeigt. Ein Repository enthält Objekte (**Object**) wie Dokumente (**Document**), Ordner (**Folder**), Beziehungen (**Relationship**) und Richtlinien (**Policy**). Dokumente und Ordner sind als Kompositum realisiert. Auf einzelne Objekte und Kompositionen von ihnen kann also auf die gleiche Weise zugegriffen werden (Gamma et al. 1994). Die eigentlichen Binärdaten hinter einem Dokument sind über einen Strom (**ContentStream**) zugreifbar. Außerdem können Dokumente und Ordner über alternative Darstellungen oder Formate (**Rendition**) verfügen. So könnte zu einem Textverarbeitungsdokument eine PDF-Version oder zu einem Bild eine qualitativ reduzierte Vorschau hinterlegt sein.

CMIS spezifiziert außerdem einen Anfragedienst, der eine relationale Sicht auf das Datenmodell bietet und Anfragen ähnlich dem SQL-92-Standard unterstützt. Die folgende Anfrage durchsucht die Dokumente nach dem Begriff „coating“ und liefert nur diejenigen, die seit dem 1. Januar 2014 erstellt wurden. Zusätzlich zur jeweiligen Kennung des Dokuments wird die Relevanz bezüglich der Volltextsuche im Ergebnis geliefert.

```
SELECT cmis:objectId, SCORE() AS relevance
FROM cmis:document
WHERE CONTAINS('coating') AND cmis:creationDate >= '2014-01-01'
ORDER BY relevance DESC
```

Das im Rahmen dieser Arbeit entwickelte PAS verwaltet prozessrelevante Dokumente nicht selbst, sondern in einem ECM-System. Die PN-Plattform integriert zu diesem Zweck eine Bibliothek für den Zugriff auf CMIS-kompatible ECM-Systeme.

Für den Zugriff auf Datenbanksysteme wird eine Schicht zur objektrelationalen Abbildung (engl. *object relational mapping*, kurz ORM) außerhalb der PN-Plattform vorausgesetzt. Hierbei werden die Tupel einer Relation auf Objekte im Adressraum der PN-Plattform abgebildet, wobei Manipulationen am Objekt dann die entsprechenden Änderungsoperationen auf der Datenbank auslösen. Da die Datenhaltung in diesem Fall für die PN-Plattform transparent geschieht, sind an dieser Stelle keine besonderen Funktionalitäten innerhalb der PN-Plattform notwendig, um mit Datenbanken kommunizieren zu können.

3.5 Organisationsverwaltung

Wie auch die Dateiverwaltung wird die Verwaltung des Organisationsmodells gemäß der Trennung von Zuständigkeiten ausgelagert (Dijkstra 1982). Zur Abbildung der Organisationsperspektive benötigt die PN-Plattform ein Modell der Organisation, auf die sich die auszuführenden Prozesse beziehen. Da auch andere Systeme einer Organisation ein solches Modell v.a. zur Authentifizierung und Autorisierung benötigen, wird diese Struktur üblicherweise in einem dedizierten System verwaltet. Mit einem solchen identitätsführenden System kann dann zum Beispiel über das *Lightweight Directory Access Protocol* (LDAP) (Wahl et al. 1997) kommuniziert werden.

Aus Sicht eines PAS setzt sich eine Organisation aus den folgenden Elementen zusammen:

- realen Aufgabenträgern, also Personen, Computer oder Computerprogramme (*Identity*),
- virtuellen Aufgabenträgern wie Rollen oder Gruppen (*Group*) und
- Beziehungen zwischen diesen (*Relation*) (Bußler 1998).

Die Schnittstelle der CPI besteht lediglich aus diesen Grundelementen, um nur minimale technische Anforderungen an ein System zur Organisationsverwaltung zu stellen. Abbildung 3.8

① zeigt diese Grundelemente und zudem eine mögliche Implementierung für die Anbindung eines LDAP-Verzeichnis (②).

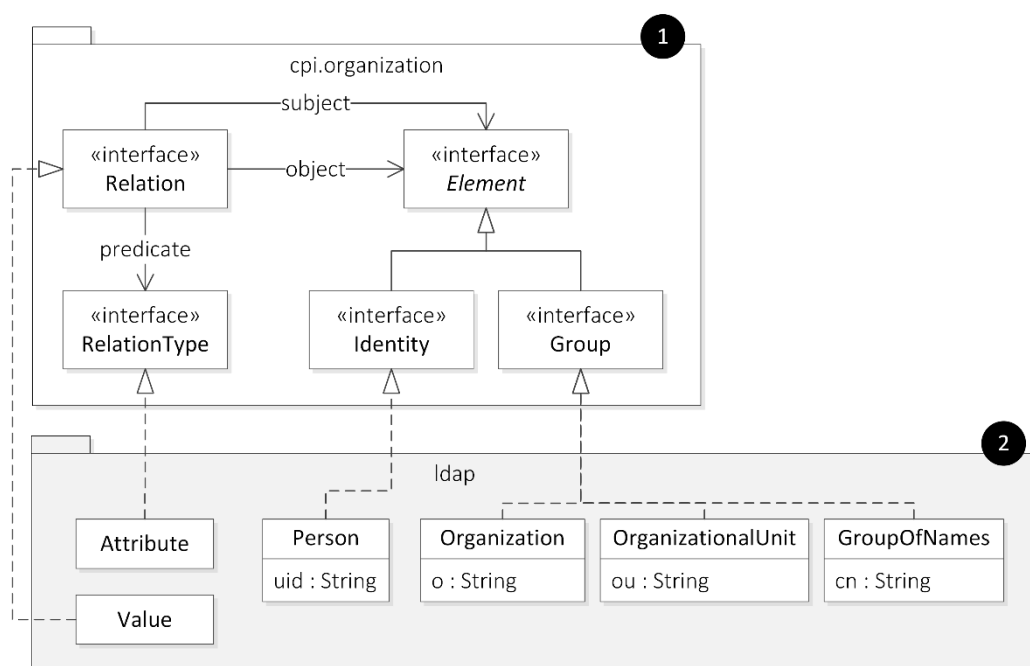


Abbildung 3.8 Schnittstelle der CPI zur Organisationsverwaltung und mögliche LDAP-Implementierung

Die von einem LDAP-kompatiblen Verzeichnisdienst verwaltete Struktur lässt sich wie in Abbildung 3.9 gezeigt als Verzeichnisbaum (engl. *directory information tree*, kurz DIT) (Zeilenga 2006) darstellen. Die Knoten des Baums bilden die Einträge des Verzeichnisses und die Kanten stellen die Ordnungsbeziehung zwischen ihnen dar. Das Beispiel zeigt einen Baum aus einer Organisation namens `acme`, die aus den zwei Einheiten `people` und `groups` besteht. In `people` befindet sich die Person mit der Benutzerkennung `michael` und in `groups` die Gruppe mit dem Namen `development`. Jedem dieser Einträge ist eine Objektklasse (engl. *object class*) zugeordnet, wobei für jede Objektklasse andere Attribute definiert sind. So sieht die Objektklasse `groupOfNames` zum Beispiel das Attribut `member` und die Objektklasse `person` die Attribute `givenName` und `sn` (von engl. *surname*).

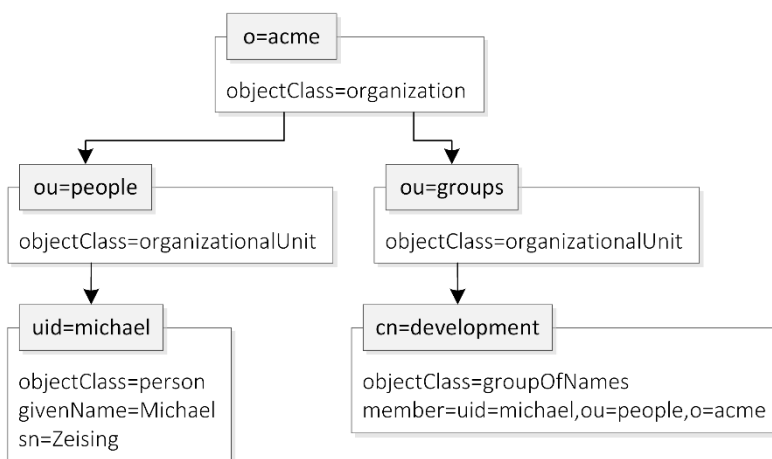


Abbildung 3.9 Exemplarischer Verzeichnisbaum eines LDAP-Verzeichnisses

Der in Abbildung 3.9 gezeigte Verzeichnisbaum lässt sich auf der Basis der in Abbildung 3.8 ② gezeigten Implementierung in das Metamodell der CPI überführen. Dabei wird für die Objektklasse **person** eine Implementierung der Schnittstelle **Person** gebildet und für die verbleibenden Objektklassen die entsprechenden Implementierungen von **Group**. Für die Ausprägung einer Beziehung und ihren Typ wurden entsprechende Implementierungen gebildet, um sie instanziierten zu können. Die Attribute, die keine spezielle Bedeutung haben oder auf eine Beziehung abgebildet werden, bilden entsprechende Klassenattribute der Implementierungen.

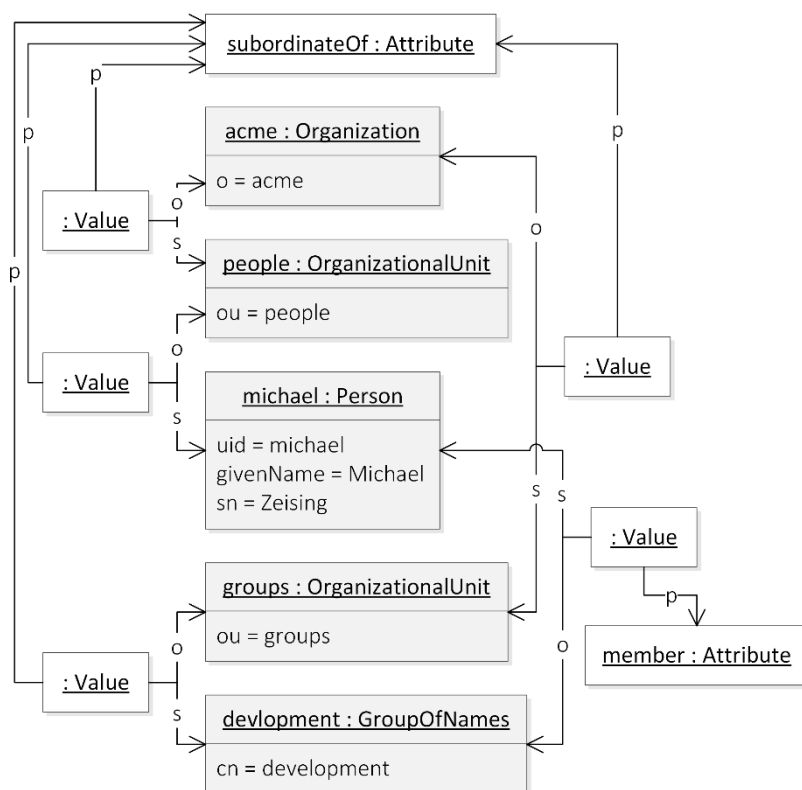


Abbildung 3.10 Entsprechung des exemplarischen Verzeichnisbaums im Rahmen der CPI (die Assoziationen „subject“, „predicate“ und „object“ sind mit „s“, „p“ und „o“ abgekürzt)

Das im LDAP-Verzeichnis verwaltete Organisationsmodell kann auf diesem Weg dem PAS zugänglich gemacht werden. Prozesse können also direkt auf das Organisationsmodell Bezug nehmen. Es kann so zum Beispiel im Prozessmodell gefordert werden, dass eine bestimmte Aktivität nur von Mitgliedern der Gruppe „development“ durchgeführt werden sollte.

3.6 Aufgabenverwaltung

Personelle Aufgaben (engl. *human tasks*) bilden die Grundlage für die Interaktion eines PAS mit menschlichen Prozessteilnehmern. Ist ein Prozessschritt als personelle Aufgabe modelliert, dann erzeugt das PAS zu dessen Ausführung eine entsprechende Aufgabeninstanz (engl. *work item*), die dann einen bestimmten Lebenszyklus durchläuft. In klassischen Architekturen bildet die Verwaltung der Aufgabeninstanzen (engl. *work item manager*) eine interne Komponente des PAS. Diese Komponente lässt sich auslagern und als Dienst betreiben. Die Schnittstelle für diesen Dienst ist unter dem Namen *WS-HumanTask* (kurz WS-HT) standardisiert worden. Sie beschreibt ein Datenmodell und entsprechende Methoden zur Verwaltung von Aufgabeninstanzen (OASIS 2010).

Da Prozessmodelle Unterscheidungen in verpflichtende und empfohlene Anteile enthalten können, muss der Dienst in der Lage sein, Bewertungen für Aktionen transportieren zu können. Außerdem soll der Zustand einer Prozessinstanz in Form der erreichten Meilensteine transportiert werden können. Die WS-HT-Schnittstelle und das entsprechende Datenmodell wurden entsprechend erweitert, um diese Eigenschaften zur Entscheidungsunterstützung realisieren zu können.

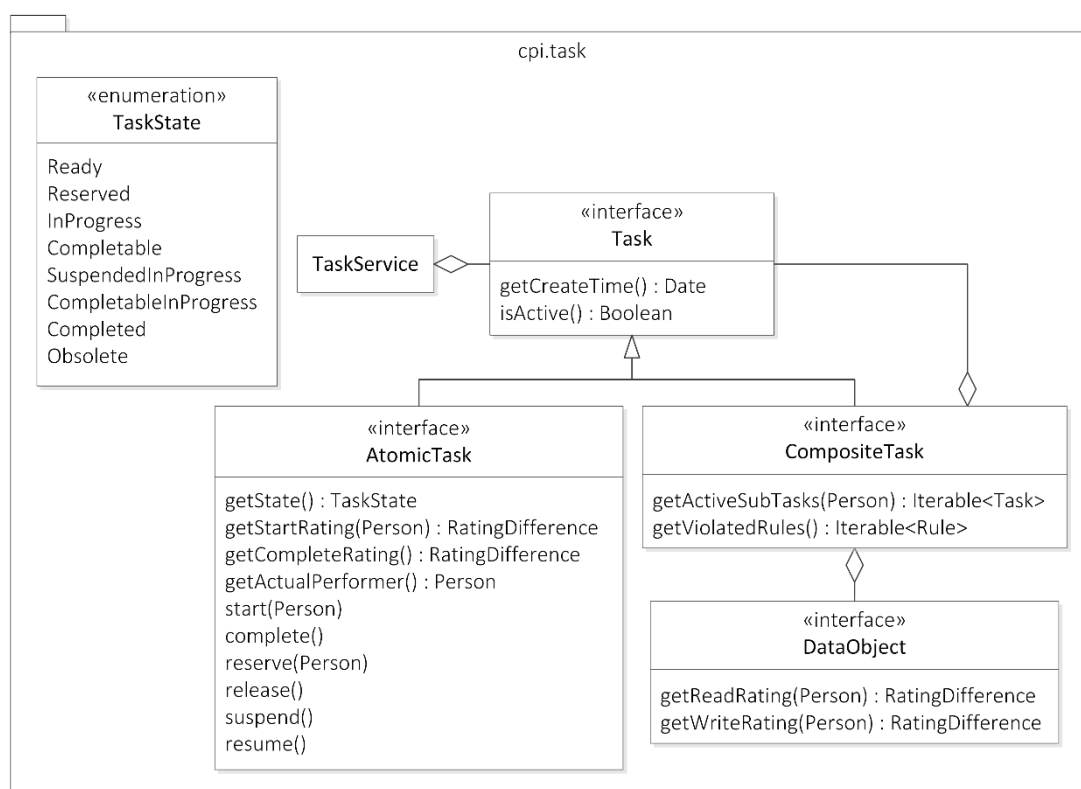


Abbildung 3.11 Schnittstelle der Aufgabenverwaltung

Die Schnittstelle ist jedoch grundsätzlich WS-HumanTask nachempfunden und kann bei Bedarf auf diese abgebildet werden. Die Informationen zur Entscheidungsunterstützung würden dabei verloren gehen.

3.7 Bereitstellung von Prozessinstanzen und Aufzeichnung von Protokollen

Um Prozessmodelle laden und Prozessinstanzen verwalten zu können, benötigt die PN-Plattform eine entsprechende Schnittstelle. Die wesentlichen Bestandteile dieser Schnittstelle sind in Abbildung 3.12 gezeigt. Die *Engine* bildet den Einstiegspunkt zur programmatischen Interaktion mit dem PAS. Sie entspricht dem *Singleton*-Muster; von ihr existiert also jeweils nur eine Instanz, die global verfügbar ist (Gamma et al. 1994).

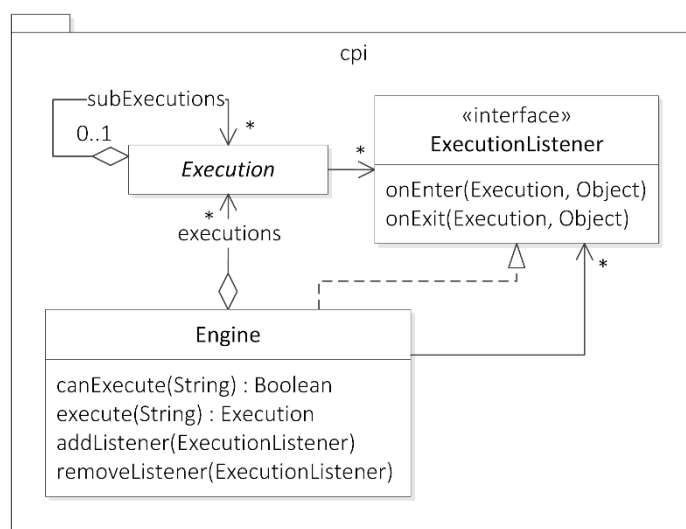


Abbildung 3.12 Schnittstelle zur Verwaltung von Instanzen und zum Logging

Über die *Engine* können mittels der *execute*-Methode neue Modelle geladen und ausgeführt werden. Prozessmodelle werden üblicherweise in Form von Textdateien ausgetauscht. Für BPMN wurde hierzu zum Beispiel ein bestimmtes XML-basiertes Format definiert. Entsprechende textuelle Repräsentationen müssen für alle Modellierungssprachen existieren, um Modelle transportieren zu können (Object Management Group Inc. 2011; Object Management Group 2014a; OASIS 2007a). Dementsprechend nimmt die *execute*-Methode einen logischen Verweis auf eine Textdatei entgegen.

Die PN-Plattform unterstützt kombinierte Prozessmodelle aus unterschiedlichen Modellierungssprachen. Da jede Sprache ihr eigenes Format definiert, liegen solche kombinierten Modelle zunächst als Sammlung unterschiedlicher Dateien vor, die sich gegenseitig logisch referenzieren. Diese Dateien bilden eine logische Einheit, nämlich einen Gesamtprozess. Um diesen Gesamtprozess gesammelt austauschen und der PN-Plattform übergeben zu können, müssen die einzelnen Dateien in einer Container-Datei, einem sogenannten Archiv, verpackt werden. Ein weit verbreitetes Container-Format mit optionaler Kompression ist das ZIP-Format (PKWARE Inc. 2012). Die *execute*-Methode nimmt neben einzelnen Modelldateien daher auch Archive im ZIP-Format entgegen. Verweise zwischen den einzelnen Modelldateien werden beim Öffnen des Archivs als relativ zu dessen Wurzel interpretiert.

Nach dem Laden eines Modells wird ein Verweis auf den Ausführungszeiger für die oberste Modellebene gehalten. Dadurch kann über `executions` jederzeit auf die laufenden und abgeschlossenen Prozessinstanzen zugegriffen werden.

Die Aufzeichnung des Verlaufs einer Prozessinstanz (engl. *logging*) gehört zu den Aufgaben eines PAS (Reichert & Weber 2012). Sie bildet die Grundlage für Ermittlung von Kennzahlen oder für die Analyse von Prozessen (engl. *process mining*). Die CPI stellt hierfür unabhängig vom Modellierungsprinzip und von der Modellierungssprache eine entsprechende Schnittstelle zu Verfügung. Entsprechend dem Beobachtermuster (Gamma et al. 1994) kann bei einer *Engine* ein *ExecutionListener* über `addListener` registriert werden, der dann alle Ausführungszeiger einer *Engine* überwacht. Das Objekt wird benachrichtigt, sobald ein Modellelement vom Zeiger betreten oder verlassen wird. Genauere Mechanismen können von den Spezialisierungen für das jeweilige Prinzip (prozedural oder regelbasiert) oder von den Implementierungen für konkrete Sprachen bereitgestellt werden. Über diesen Mechanismus kann zum Beispiel eine Aufzeichnung nach dem XES-Standard (Günther 2009) realisiert werden.

3.8 Webbasierte Teilnehmeroberfläche

Ziel der Arbeit ist die Unterstützung von Prozessen durch ein IT-System. Dazu müssen personelle Aufgaben den entsprechenden Prozessteilnehmern zugewiesen bzw. angeboten werden und prozessrelevante Daten zur Verfügung gestellt werden (Reichert & Weber 2012). Diese Interaktion mit den menschlichen Prozessteilnehmern wird durch eine Teilnehmeroberfläche abgedeckt. Diese Oberfläche soll ausschließlich die Schnittstellen der CPI für die Aufgabenverwaltung und das Bereitstellen von Prozessmodellen nutzen.

Software als Webanwendung bereitzustellen ist zum de facto-Standard der Industrie geworden. Unternehmen wie Google haben sich durch Webanwendungen überhaupt erst etabliert und setzen nach wie vor auf ein entsprechendes Produktportfolio. Auch Anwendungen wie die Sammlung der Office-Programme von Microsoft, die ursprünglich als Desktopanwendungen entworfen wurden, werden derzeit in Microsoft Office Online portiert. Entsprechend dieser bewährten Vorgehensweise ist auch die Teilnehmeroberfläche des PAS als Webanwendung konzipiert. Der im Rahmen dieser Arbeit entwickelte Prototyp setzt auf der Java EE-Technologie auf.

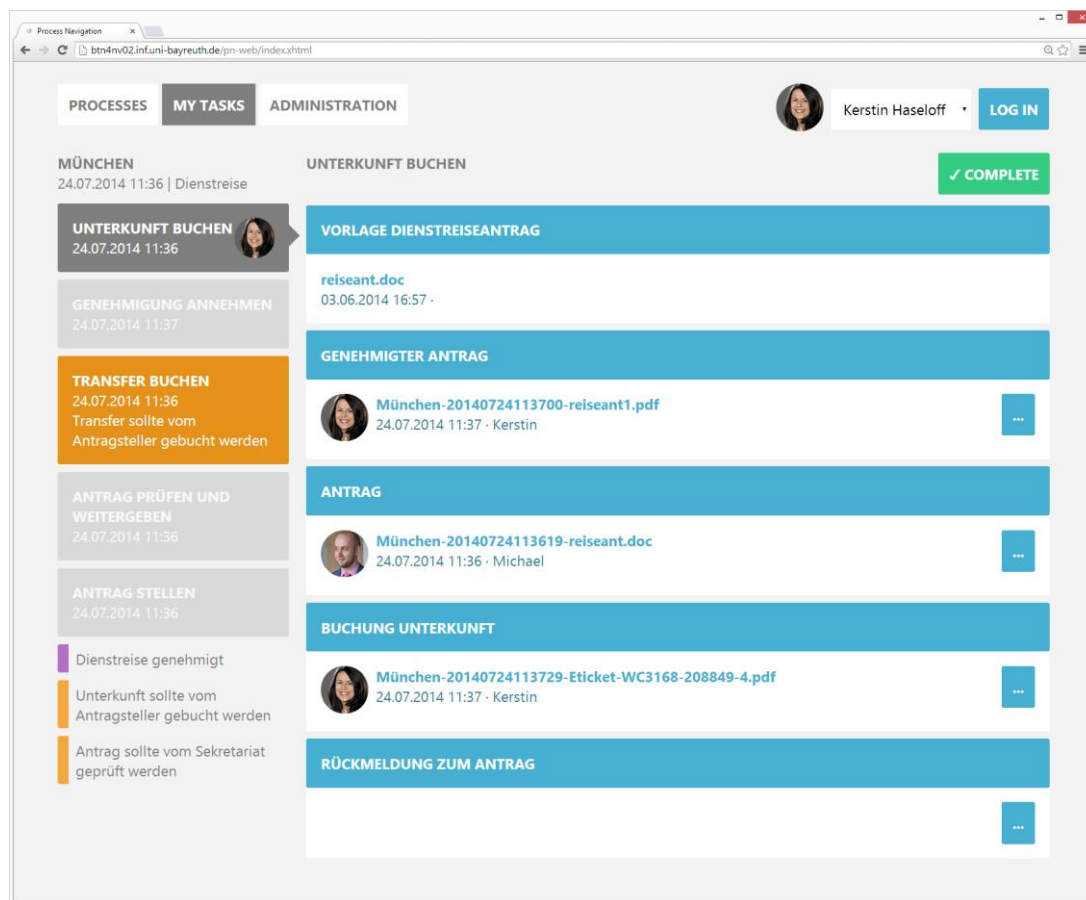


Abbildung 3.13 Webbasierte Teilnehmeroberfläche des PAS

Abbildung 3.13 zeigt die Oberfläche für Teilnehmer eines Prozesses. Die Ansicht ist in zwei Bereiche untergliedert. Der linke Bereich zeigt für jede laufende Prozessinstanz die entsprechenden Aufgaben an. Die Datenobjekte der laufenden Instanzen werden im rechten Bereich angezeigt.

3.8.1 Aufgabenansicht

Die Aufgabenansicht nimmt das linke Drittel der Ansicht in Abbildung 3.13 ein. Hier werden für jede laufende Prozessinstanz der Name (im Beispiel „München“), der Startzeitpunkt und der Name des zugrundeliegenden Prozessmodells (im Beispiel „Dienstreise“) angezeigt. Für jede Aufgabe wird wiederum der Name der Aufgabe (zum Beispiel „Unterkunft buchen“) entsprechend dem Prozessmodell und der Zeitpunkt zu dem die Aufgabe zum ersten Mal angeboten wurde angezeigt. Wird die Aufgabe momentan bearbeitet, dann wird das Profilbild des aktuellen Bearbeiters angezeigt.

3.8.2 Datenansicht

Die Datenansicht nimmt die rechten zwei Drittel der Ansicht in Abbildung 3.13 ein. Hier werden die Datenobjekte angezeigt, auf die im Zusammenhang mit der aktuell bearbeiteten Aufgabe zugegriffen werden kann. Für jedes Datenobjekt wird sein Name und das aktuell hinterlegte Dokument bzw. der aktuelle Datenwert zusammen mit dem Zeitpunkt der Erstellung und dem

Autor angezeigt. Außerdem können neue Datenwerte angelegt beziehungsweise Dokumente hochgeladen werden.

3.8.3 Entscheidungsunterstützung

Die verschiedenen Modalitäten werden auf der Oberfläche durchgängig mit Hilfe der in Tabelle 3.1 gezeigten Farben gekennzeichnet. Durch die farbliche Kodierung erhält die Oberfläche einen unterstützenden Charakter und gibt Hinweise bei der Bearbeitung des Prozesses.

Tabelle 3.1 Farbliche Kodierung der Weboberfläche

Farbe	Bedeutung
● hellgrau	Nicht erlaubt
● orange	Nicht empfohlen
● blau	Neutral
● grün	Empfohlen
● lila	Erreichter Meilenstein

Zudem werden unterhalb der Aufgaben für jede Prozessinstanz die nicht beachteten Empfehlungen und die erreichten Meilensteine aufgezählt. Für jede nicht empfohlene Aktion werden die Regeln angezeigt, die verletzt werden würden (zum Beispiel „Transfer sollte vom Antragsteller gebucht werden“ bei Aufgabe „Transfer buchen“).

3.9 Zusammenfassung und Bewertung

Das zentrale Konzept der CPI ist die Unterstützung von verschachtelten und vor allem von kombinierten Prozessmodellen, also Modellen, die sich aus verschiedenen Sprachen und sogar verschiedenen Prinzipien zusammensetzen. Außerdem bietet sie die Grundlage für den Transport von Bewertungen und Regeln, die vor allem bei der Ausführung von regelbasierten Modellen von Bedeutung sind. Die CPI integriert zudem die für das PAS notwendigen Dienste wie die Datei-, Organisations- und Aufgabenverwaltung. Daneben bietet die CPI zentrale Schnittstellen für die Verwaltung von Prozessen und die Aufzeichnungen ihrer Abläufe.

Die gemeinsame Prozessinfrastruktur (engl. *Common Process Infrastructure*, kurz CPI) bildet die Grundlage für die Ausführung von Prozessmodellen unabhängig vom zugrundeliegenden Modellierungsprinzip. Die Ausführungskomponenten für sowohl prozedurale als auch regelbasierte Modelle setzen auf der CPI auf und basieren auf ihren Bausteinen und Konzepten. Die CPI selbst hat jedoch keine Kenntnis vom jeweiligen Prinzip oder gar einer konkreten Modellierungssprache. Mit dem Entwurf der CPI und auf ihr aufbauenden Komponenten ist es erstmals gelungen, einen Großteil der Funktionalität eines PAS von Modellierungsprinzip und -sprache unabhängig zu machen.

4 Prozedurale Prozess-VM (PPVM)

Die in Abschnitt 3 beschriebene *Common Process Infrastructure* (CPI) bildet die Grundlage für die Ausführung sowohl prozeduraler, als auch regelbasierter Prozessmodelle. Da sich die Ausführung dieser beiden Prinzipien grundsätzlich unterscheidet, sind jedoch unterschiedliche Spezialisierungen der CPI notwendig. Um prozedurale Sprachen wie BPMN auf der PN-Plattform ausführen zu können, werden die generischen Konzepte der CPI im Folgenden zunächst für prozedurale Sprachen spezialisiert (vgl. Abbildung 4.1).

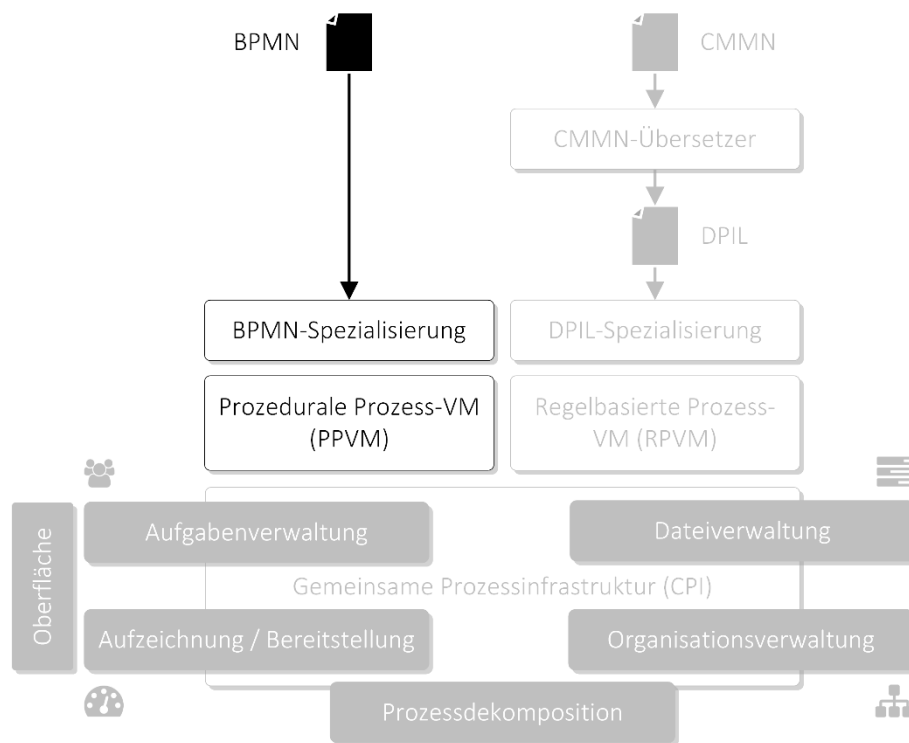


Abbildung 4.1 Aufbau der PN-Plattform mit PPVM

Zur Interpretation prozeduraler Modelle eignet sich, wie bereits erwähnt, die Simulation von Marken, die sich durch den Prozessgraphen bewegen. Die PVM nach Baeyens und Faura stellt eine Realisierung dieses Konzepts dar. Um die CPI nun für prozedurale Modelle zu erweitern, werden einige der beim Entwurf der CPI aus der PVM entfernten Spezifika für prozedurale Modelle nun wieder hinzugefügt. Die *Procedural Process Virtual Machine* (kurz PPVM) stellt diese Spezialisierung der CPI für prozedurale Modelle dar. Ihre Kernelemente sind in Abbildung 4.2 dargestellt.

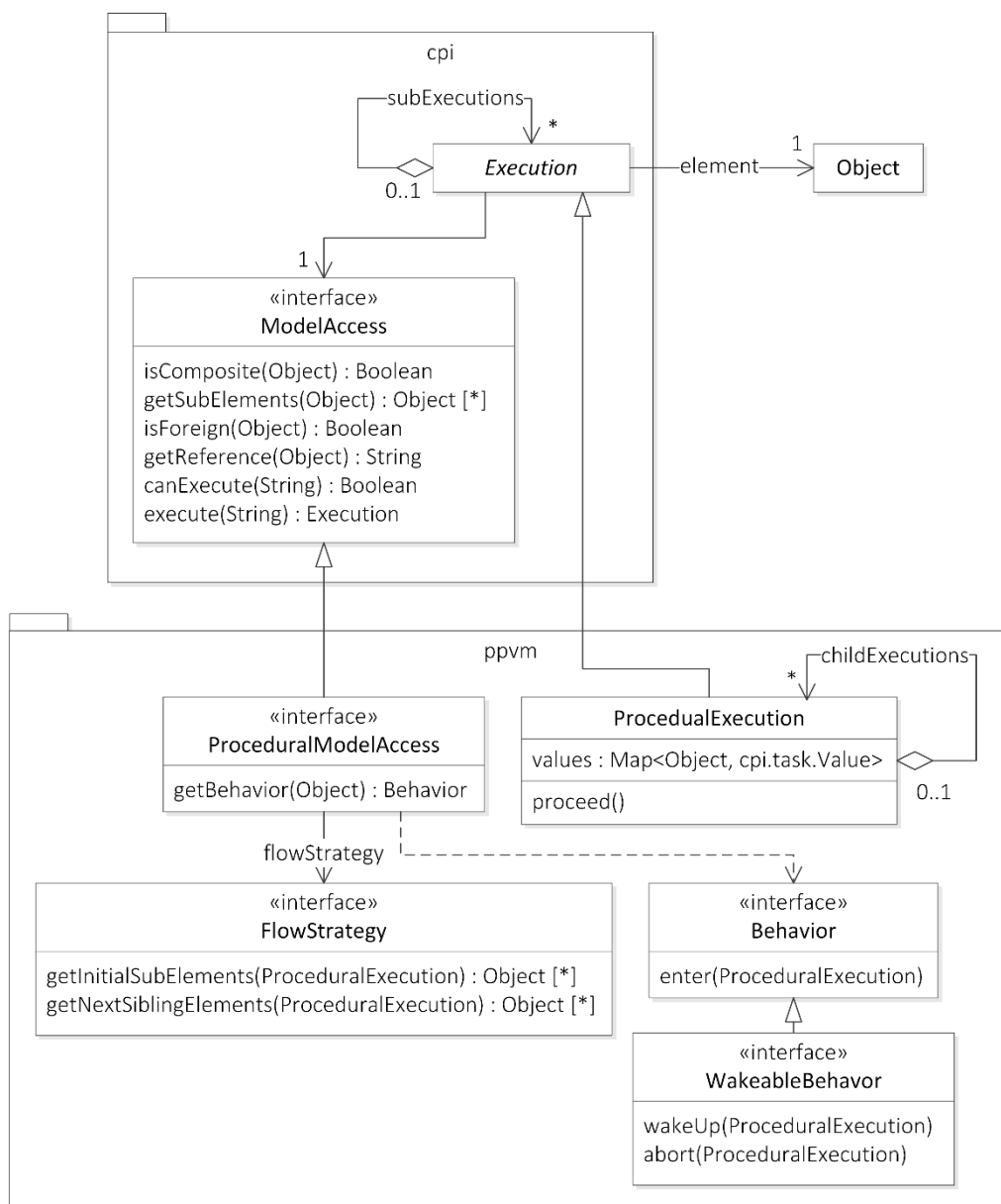


Abbildung 4.2 Kernelemente der PPVM und ihr Zusammenhang mit der CPI

Der Ausführungszeiger für ein prozedurales Modell (**ProceduralExecution**) erweitert den Ausführungszeiger (**Execution**) der CPI und erbt damit alle seine Eigenschaften. Er verweist also ebenso auf ein aktuelles Modellelement (**element**), kann untergeordnete Zeiger für Subprozesse enthalten (**subExecutions**) und verfügt über ein Objekt für den sprachspezifischen Modellzugriff (**ModelAccess**). Die **ProceduralExecution** definiert zusätzlich zur **Execution** die Methode **proceed**, die den Zeiger innerhalb des Modells weiterrücken lässt. Die zusätzliche Assoziation **childBranches** referenziert aufgesplante Ausführungszweige. Verzweigt sich eine Ausführung zum Beispiel an einem parallelen Gatter (AND-Verzweigung), wird ein weiterer Ausführungszeiger erzeugt und dem Hauptzweig untergeordnet. An einer Synchronisation (AND-Zusammenführung) können diese Zweige dann wieder zusammengeführt werden. Die PPVM ist damit in der Lage parallele Ausführungszweige zu verfolgen und zu synchronisieren.

Ein Datenobjekt muss in verschiedenen Prozessinstanzen und in verschiedenen parallelen Ausführungspfaden verschiedene Werte annehmen können. Dies wird erreicht, indem man die aktuellen Datenwerte im Ausführungszeiger verwaltet. Bereits Baeyens und Faura haben beim Entwurf der PVM festgestellt, dass der prozedurale Ausführungszeiger einen natürlichen Geltungsbereich für die Datenwerte einer Prozessinstanz bildet (Baeyens & Faura 2007). Der Zeiger verknüpft dazu im Attribut `values` über ein assoziatives Datenfeld (`Map`) die Datenobjekte des Prozessmodells mit deren aktuellem Datenwerte. Ein Datenwert muss die Schnittstelle `Value` der CPI implementieren, damit er über die Aufgabenverwaltung transportiert werden kann. Der Typ des modellierten Datenobjekts hängt von der jeweiligen Prozessmodellierungssprache ab und kann daher auf der Ebene der PPVM nicht festgelegt werden (Wurzeltyp `Object`). Die PPVM ist hiermit in der Lage, die Werte von Datenobjekten bezogen auf einen Ausführungszeiger zu verwalten.

Wird ein neuer Subprozess betreten oder eine Aktivität beendet, dann müssen die ersten bzw. die nächsten Knoten im Prozessgraphen bestimmt werden, um die Ausführung fortzuführen. Die konkrete Vorgehensweise soll nicht wie durch die PVM nach Baeyens und Faura vorgeschlagen fest implementiert, sondern an die jeweilige Modellierungssprache delegiert werden. Sie ist dadurch austauschbar und vor der PPVM verborgen (vgl. Strategie-Entwurfsmuster (Gamma et al. 1994)). Auch der sprachspezifische Modellzugriff (`ModelAccess`) wird daher durch die PPVM erweitert (`ProceduralModelAccess`). Der Modellzugriff für prozedurale Sprachen verweist zusätzlich auf eine `FlowStrategy`, die von der jeweiligen Modellierungssprache implementiert werden muss. Die Methode `getInitialSubElements` liefert für einen gegebenen Zeiger die Elemente, die nach dem Start des Prozesses betreten werden sollen, also zum Beispiel alle Aktivitäten ohne eingehende Sequenzflüsse. Die Methode `getNextSiblingElements` liefert für einen Zeiger alle Elemente, die von ihm aus als nächstes betreten werden sollen, also zum Beispiel alle Elemente, die über einen direkten ausgehenden Sequenzfluss mit einem parallelen Gatter verbunden sind (AND-Verzweigung). Damit unterstützt die PPVM Konstrukte zur Modellierung von Kontrollflüssen wie den Sequenzfluss in BPMN unabhängig von einer konkreten Modellierungssprache.

Das konkrete Verhalten eines Prozessknoten wird analog zur PVM nach Baeyens und Faura ebenfalls vollständig vor der PPVM verborgen. Für jeden Knotentyp muss ein Verhalten (`Behavior`) implementiert werden. Wird ein Knoten durch den Ausführungszeiger betreten, so wird das dem Knotentyp entsprechende Verhalten erzeugt (zum Beispiel der Aufruf eines Dienstes) und die Methode `enter` aufgerufen. Soll der Knoten verlassen werden, muss die Methode `proceed` des Zeigers aufgerufen werden. Für Knotentypen die einen Wartezustand erlauben wie zum Beispiel eine personelle Aufgabe muss eine `WakeableBehavior` implementiert werden und die Methode `wakeUp` zum Beispiel von der Aufgabenverwaltung aufgerufen werden, um den Wartezustand zu verlassen. Auch bei der Implementierung des Verhaltens von Prozessknoten bleibt die PPVM also unabhängig von einer konkreten Modellierungssprache.

4.1 Ausdruckssprache

Eines der meistgenutzten Elemente in prozeduralen Modellen ist die exklusive Verzweigung (auch „Entscheidung“) (zur Muehlen & Recker 2008). Diese Art der Verzweigung wird verwendet, um

auszudrücken, dass die Ausführung des Prozesses nur einem der nachfolgenden Pfade folgt. Soll der Prozess maschinell interpretiert, also zum Beispiel von einem PAS ausgeführt werden, so müssen die Bedingungen für die ausgehenden Pfade einer exklusiven Verzweigung vom PAS getroffen, also ebenso maschinell interpretiert werden können.

Das exemplarische Prozessmodell in Abbildung 4.3 enthält eine exklusive Verzweigung von der ein bedingter Fluss ❶ (Raute am Pfeilanzug) und der Vorgabefluss (engl. *default flow*) (Querstrich am Pfeilanzug) ausgehen. Die Bedingung „Kreditrahmen eingehalten?“ ist lediglich eine Beschriftung, die nur für das Diagramm relevant ist. In Teil ❷ ist das ausführliche BPMN-Modell zu Sequenzfluss ❶ gezeigt. Um die Bedingung für den Fluss formalisieren zu können, bietet BPMN die Möglichkeit einen Ausdruck in einer formalen Sprache zu hinterlegen. Im Beispiel wurde der Ausdruck „Summe.value <= 20000“ hinterlegt. Dieser Ausdruck kann vom PAS bei der Aktivierung der Verzweigung ausgewertet werden und gibt an, ob der jeweilige Fluss gewählt werden soll oder nicht. BPMN lässt dabei jedoch offen, welche Sprache an dieser Stelle verwendet wird.

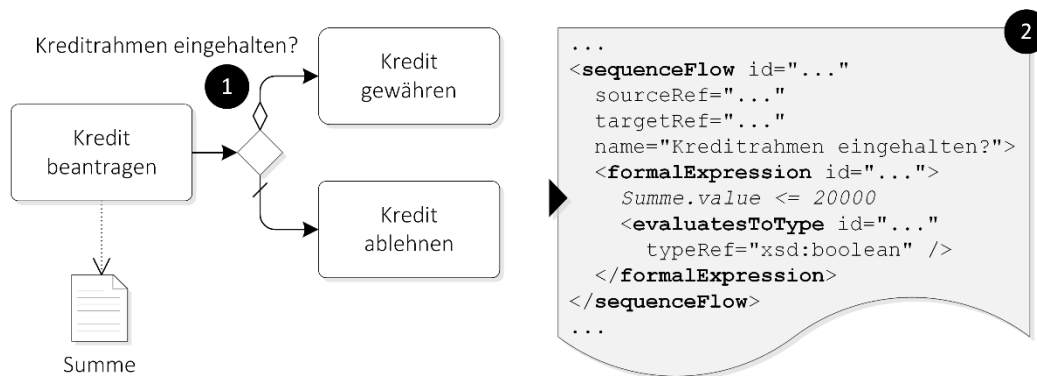










Abbildung 4.3 Beispiel für Ausdrücke in BPMN-Modellen



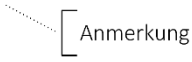
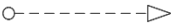


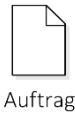

Die *Expression Language* (EL) der Java EE-Plattform bietet sich als einfache Ausdruckssprache an. Die Sprache bietet eine einfache Syntax für den Zugriff auf Objekte, für Vergleiche und für arithmetische Operationen. Die Auflösung der Objekte, Funktionen und Variablen geschieht über eine festgelegte Schnittstelle. Die EL lässt sich damit leicht in beliebige Umgebungen integrieren (Oracle Corporation 2013). Die PN-Plattform enthält die Implementierung *Java Unified Expression Language* (JUEL), um EL-Ausdrücke in Prozessmodellen auswerten zu können.

4.2 Integration von BPMN

In (zur Muehlen & Recker 2008) wurden 120 repräsentative BPMN-Modelle gesammelt, die entweder im Internet verfügbar sind, aus Schulungsunterlagen stammen oder von Beratern verwendet wurden. Diese Modelle wurden hinsichtlich der Verwendung von Modellierungskonstrukten untersucht. Das Ergebnis der Studie ist, dass vor allem die in Tabelle 4.1 aufgezählten Elemente zur Modellierung eingesetzt werden.

Tabelle 4.1 Häufig verwendete BPMN-Elemente

Element	Diagramm	Bedeutung	Ver- wen- dung	Relevant ?
Sequenzfluss (engl. <i>sequence flow</i>)		Der Sequenzfluss zeigt die Reihenfolge von Elementen innerhalb eines Prozesses an.	100%	●
Aufgabe (engl. <i>task</i>)		Eine Aufgabe ist eine Aktivität innerhalb eines Prozesses, die sich nicht weiter aufteilen lässt.	96%	●
Startereignis (engl. <i>start event</i>)		Das Startereignis zeigt an, wo ein Prozess und damit sein Fluss beginnt.	62%	●
Nachrichtenbasiertes Zwischenereignis (engl. <i>message intermediate event</i>)		Ein Zwischenereignis zeigt an, dass etwas zwischen dem Start und dem Ende eines Prozesses geschieht.	20%	●
Endereignis (engl. <i>end event</i>)		Das Endereignis zeigt an, wo ein Prozess und damit sein Fluss endet.	66%	●
Terminierendes Endereignis (engl. <i>terminate end event</i>)		An einem terminierenden Endereignis werden alle Aktivitäten des Prozesses sofort beendet.	21%	●
Pool		Ein Pool stellt den Teilnehmer einer Zusammenarbeit (engl. <i>collaboration</i>) dar und kann einen Prozess enthalten.	56%	
Lane		Eine Lane ist eine Unterteilung eines Prozesses bzw. eines Pools um die entsprechenden Aktivitäten zu gruppieren.	35%	

Exklusive Gatter (engl. <i>exclusive gateway</i>)		Ein exklusives Gatter zeigt alternative Pfade innerhalb eines Prozesses an.	52%	●
Paralleles Gatter (engl. <i>parallel gateway</i>)		Ein paralleles Gatter erzeugt parallele Pfade innerhalb eines Prozesses oder führt sie wieder zusammen.	30%	●
Anmerkung (engl. <i>annotation</i>)		Eine Anmerkung beinhaltet zusätzliche Informationen für den Leser eines Diagramms.	35%	
Nachrichtenfluss (engl. <i>message flow</i>)		Der Nachrichtenfluss zeigt den Austausch von Nachrichten zwischen den Teilnehmern einer Zusammenarbeit an.	33%	●
Nachrichtenbasiertes Starterereignis (engl. <i>message start event</i>)		Der Prozess wird durch den Erhalt einer Nachricht von einem anderen Teilnehmer gestartet.	42%	●
Subprozess		Ein Subprozess ist eine Aktivität innerhalb eines Prozesses, deren Aufbau modelliert wurde.	23%	●
Datenobjekt		Ein Datenobjekt ist eine physische oder informationelle Einheit, die innerhalb eines Prozesses erzeugt, manipuliert oder genutzt wird.	23%	●
Datenverbindung (engl. <i>data association</i>)		Datenverbindungen zeigen die Bewegung von Daten zwischen Quellen und Senken und kapseln eventuelle Transformationen.	25%	●

Über die Hälfte der Modelle bestehen aus Start- und Endereignissen, Aufgaben, Sequenzflüssen, exklusiven Verzweigungen und Pools. Die Elemente der informationsbezogenen Perspektive, nämlich Datenobjekt und –verbindung werden nur in etwa einem Viertel der Modelle verwendet. Alle Konstrukte, die in Tabelle 4.1 nicht aufgeführt sind, wurden in weniger als 20% der Modelle verwendet.

Um also entsprechend dem Paretoprinzip (Pareto & Schwier 1971) 80% der Modelle unterstützen zu können, müssen die für die Ausführung relevanten Elemente aus Tabelle 4.1 durch die PN-Plattform interpretiert werden können. Die Realisierung der 13 resultierenden Elemente wird im Folgenden beschrieben.

4.2.1 Sequenzflüsse, Gatter, Ereignisse und Nachrichten

Die `FlowStrategy` für BPMN interpretiert den Sequenzfluss. Für einen neu betretenen Prozess werden dazu alle Elemente ohne eingehenden Sequenzfluss ausgewählt. Zur Bestimmung der nächsten Knoten werden alle ausgehenden Sequenzflüsse betrachtet. Ist die jeweilige Bedingung erfüllt oder keine angegeben, wird der entsprechende Zielknoten zurückgegeben.

Außerdem wird für jeden Gattertyp (`ExclusiveGateway`, `InclusiveGateway`, `ParallelGateway`) das entsprechende Verhalten hinterlegt. Dabei muss jeweils das zusammenführende und aufspaltende Verhalten implementiert werden, also zum Beispiel sowohl die Synchronisation als auch die Aufspaltung nebenläufiger Pfade am parallelen Gatter.

Das gewöhnliche Starterereignis ist optional. Wie auch an der jeder Aktivität ohne eingehende Sequenzflüsse wird an jedem Starterereignis ein Ausführungszeiger erzeugt und aktiviert. An einem nachrichtenbasierten Starterereignis hingegen wird ein inaktiver Ausführungszeiger erzeugt, der erst durch die entsprechende Nachricht aktiviert wird. Auch an einem nachrichtenbasierten Zwischenereignis wird der Zeiger deaktiviert und die Ausführung erst nach Eingang der Nachricht fortgesetzt.

Ein gewöhnliches Endereignis ist optional. Wie auch im Fall, dass eine Aktivität keine ausgehenden Sequenzflüsse besitzt, wird der entsprechende Ausführungszeiger beendet. Wurden alle Zeiger einer Prozessinstanz beendet, so wird auch ihr Zeiger beendet. Erreicht ein Zeiger ein terminierendes Endereignis, so werden alle Zeiger der Prozessinstanz sofort beendet und ihre jeweiligen Elemente (zum Beispiel Aktivitäten) abgebrochen. Damit wird auch die Prozessinstanz beendet.

4.2.2 Prozesse und Aufgaben

Die für BPMN spezifische Implementierung der `ModelAccess`-Schnittstelle (siehe Abschnitt 3.2.3) unterstützt die Verschachtelung durch eingebettete Subprozesse (`SubProcess`). Von den Aufgabentypen werden exemplarisch die personelle Aufgabe (`UserTask`) und der Skriptaufruf (`ScriptTask`) realisiert und entsprechende Verhalten hinterlegt. Das Verhalten der personellen Aufgabe erzeugt eine entsprechende Aufgabeninstanz in der Aufgabenverwaltung der CPI und führt die Ausführung fort, sobald die Instanz abgeschlossen wird. Pools kapseln den Teilprozess einer beteiligten Organisation und die Bedeutung von Lanes ist nicht festgelegt. Für die Zuweisung

von Teilnehmern zu Aufgaben werden daher Konstrukte verwendet, die keine Entsprechung im Diagramm haben (Object Management Group Inc. 2011).

4.2.3 Datenobjekte und -verbindungen

Datenobjekte (**DataObject**) dienen in BPMN zur Modellierung jeglicher Informationseinheiten, die während des Prozesses erzeugt, verändert oder genutzt werden. Informationen zum Typ eines Datenobjekts werden in der entsprechenden Definition (**ItemDefinition**) gekapselt (Object Management Group Inc. 2011). Im Rahmen dieser Arbeit werden zwei grundlegende Definitionen und damit Typen von Datenobjekten umgesetzt:

Variable: Datenobjekte vom Variablentyp enthalten einen Verweis auf ein Objekt im Adressraum des PAS. Hier können Werte wie Zahlen und Zeichenketten aber auch Objekte eines Domänenmodells wie ein Kunden- oder Auftragsobjekt referenziert werden. Der Lebenszyklus ist damit an den der Prozessinstanz gebunden. Am Ende einer Prozessinstanz gehen die Datenobjekte vom Typ Variable verloren.

Dokument: Datenobjekte vom Typ Dokument enthalten einen Verweis auf ein Dokument in einem CMIS-kompatiblen ECM-System, also auf beliebige außerhalb des PAS verwaltete Dateien. Der Lebenszyklus von Dokumenten ist damit prinzipiell unabhängig von dem einer Prozessinstanz, wobei der Verweis auf das Dokument mit dem Ende der Instanz ungültig wird. Zudem kann für Dokumente je nach angebundenem ECM-System eine Revisionsverwaltung realisiert werden.

4.3 Zusammenfassung

Die Prozedurale Prozess-VM (PPVM) bildet die Grundlage für die Ausführung prozeduraler Prozessmodelle. Das zentrale Konzept der PPVM bildet die Verfolgung paralleler Ausführungspfade innerhalb eines Modells. Darauf aufbauend verwaltet sie die Werte der Datenobjekte eines Prozessmodells. Für die Realisierung von Verzweigungen in Prozessmodellen ist die Auswertung einfacher Ausdrücke notwendig. Diese wird ebenfalls auf der Ebene der PPVM durchgeführt. Die entsprechenden Details bleiben einer Spezialisierung für eine konkrete Modellierungssprache damit verborgen. Die PPVM stellt lediglich eine Reihe von Schnittstellen zur Verfügung, durch deren Implementierung die Ausführung einer konkreten Modellierungssprache umgesetzt werden kann.

5 Regelbasierte Prozess-VM (RPVM) – Ereignisbasierte Ausführung von Prozessen

Die oben beschriebene PPVM bildet die Grundlage für die Ausführung von Sprachen wie BPMN. Sie führt prozedurale Modelle effizient aus, indem sie die Bewegung von Marken beziehungsweise Ausführungszeigern entlang des Kontrollflusses simuliert. Der Zustand eines regelbasierten Modells ändert sich hingegen nicht entlang eines Kontrollflusses, sondern entsprechend den Regeln des Prozesses. Regelbasierte Modelle lassen sich daher nicht durch die Simulation von Marken sondern auf der Basis von *Ereignissen* ausführen.

Jedes Element eines Prozesses, sei es eine Aktivität oder ein Datenobjekt, besitzt einen bestimmten Lebenszyklus, den es durchläuft. Eine personelle Aufgabe zum Beispiel wird einem Teilnehmer zugewiesen, gestartet, unterbrochen, fortgesetzt oder beendet. Der Wert eines Datenobjekts wird erzeugt, verändert oder gelesen. Diese elementaren Ereignisse bilden die Grundbausteine einer ereignisbasierten Ausführung von Prozessen. Die Ausführung eines regelbasierten Prozessmodells kann als Simulation eines Zeitschritts in die Zukunft realisiert werden. Wird die Ausführung eines regelbasierten Prozesses *ausgelöst*, so werden für jedes Element des Modells (Aktivitäten, Datenobjekte usw.) die jeweils nächsten möglichen Ereignisse *generiert*. Zusammen mit dem bisherigen *Ereignisverlauf* entsteht so für jedes dieser möglichen nächsten Ereignisse ein möglicher zukünftiger Verlauf. Diese simulierten Verläufe werden anschließend auf der Grundlage der Prozessregeln *bewertet*. Die jeweilige Änderung gegenüber der letzten Bewertung wird dann durch das System *behandelt* und entsprechende Aktionen werden durchgeführt. Wird für eine personelle Aufgabe zum Beispiel der Beginn durch einen bestimmten Teilnehmer möglich, so führt das System eine entsprechende Zuweisung der Aufgabe zu einem Teilnehmer durch.

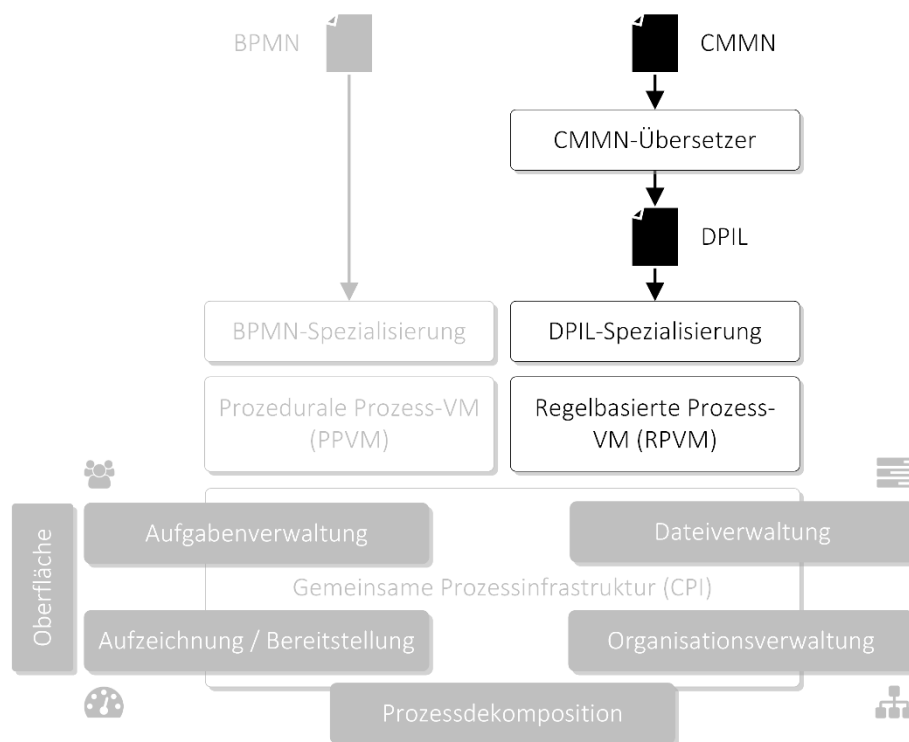


Abbildung 5.1 Aufbau der PN-Plattform mit RPVM

Die *Rule-based Process Virtual Machine* (RPVM) spezialisiert die gemeinsame Prozessinfrastruktur (CPI) der PN-Plattform (vgl. Abbildung 5.1). Sie bildet die Grundlage für die Ausführung regelbasierter Prozessmodelle auf der Basis von Ereignissen und realisiert das oben skizzierte Ausführungsprinzip. Die vier Stufen *Auslösung*, *Generierung*, *Bewertung* und *Behandlung* sind dabei unabhängig von konkreten Ereignissen, und damit auch unabhängig von einer konkreten Modellierungssprache. Änderungen an einer regelbasierten Modellierungssprache, wie zum Beispiel neue Perspektiven, erfordern somit keine Änderungen an der RPVM. Diese bisher einzigartige Architektur wird im Folgenden näher betrachtet.

5.1 Auslöser

Die regelbasierte Ausführung wird auf einen Auslöser hin fortgeführt. Erst nach diesem Auslöser werden die nächsten möglichen Ereignisse simuliert und Aktionen durchgeführt. Die einfachste Form ist die Auslösung durch ein Ereignis der Prozessinstanz. Die Ausführung wird also fortgeführt, sobald im laufenden Prozess ein Ereignis eintritt, also zum Beispiel eine Aktivität abgeschlossen oder ein Datenobjekt verändert wurde. Das PAS und die Prozessinstanz reagieren damit ausschließlich auf externe Ereignisse und bilden somit ein reaktives System. Dieser Modus ist zulässig, solange das Prozessmodell ein abgeschlossenes System bildet, also nicht auf Zusammenhängen basiert, die sich ohne Eintreten eines Ereignisses ändern können. Eine erneute Simulation ohne Eintreten eines Ereignisses würde zur selben Bewertung der Ereignisse und damit zu keinen Aktionen durch das System führen. Entsprechende Prozessmodelle werden fortan als reaktive (engl. *reactive*) Modelle bezeichnet.

Hängt das Prozessmodell jedoch von Zusammenhängen ab, die nicht durch explizite Ereignisse markiert werden, wie zum Beispiel von der Zeit oder von Daten, die außerhalb des PAS verwaltet werden, dann kann die Ausführung nicht mehr nur nach Ereignissen des Prozesses ausgelöst werden. In diesem Fall muss die Ausführung auch mit unverändertem Ereignisverlauf nach bestimmten Zeitintervallen fortgeführt werden. Entsprechende Prozessmodelle werden fortan als aktive (engl. *active*) Modelle bezeichnet.

5.2 Erzeugung von Ereignissen

Ein *Ereignis* ist ein eindeutig identifizierbares Objekt, das sich auf ein Element eines Prozessmodells bezieht (*element*). Die RPVM ist unabhängig von einer konkreten Modellierungssprache und hat daher keine Kenntnis von den Modellelementen der ausgeführten Sprache. Der Typ des Modellelements eines Ereignisses kann daher nicht eingeschränkt werden (*Object* bildet die Wurzelklasse). Da in einer Modellierungssprache die Ereignisse meist eine zeitliche Ordnung aufweisen, besitzt ein Ereignis grundsätzlich einen Zeitstempel (*time*).

Die Verwaltung zusammengesetzter Prozessmodelle wird durch das Konzept der verschachtelten Ausführungszeiger bereits durch die CPI abgedeckt. Die RPVM macht diesen Aspekt regelbasierten Modellen zugänglich, indem sie für das Betreten und Abschließen eines zusammengesetzten Prozesses die entsprechenden Ereignisse *Enter* und *Exit* vorsieht. Das Ereignis *Enter* tritt ein und wird an den Verlauf einer Instanz angehängt, sobald eine zusammengesetzte Aktivität innerhalb der Instanz betreten wird. Dementsprechend tritt *Exit* ein und wird angehängt, sobald sie wieder verlassen wird. Die Regeln des Prozessmodells können sich somit auf die Ereignisse zusammengesetzter Aktivitäten beziehen und das Konzept muss bei der Spezialisierung für eine konkrete Modellierungssprache nicht mehr behandelt werden.

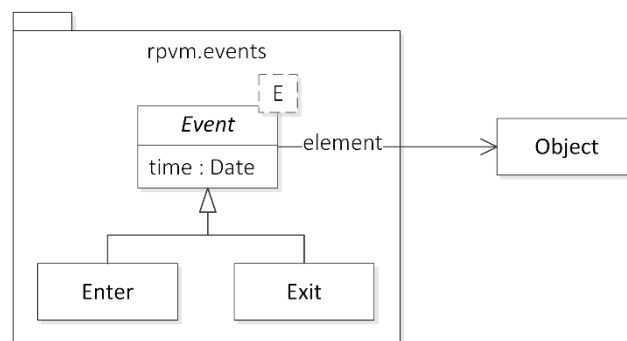


Abbildung 5.2 Die durch die RPVM definierten Basisklassen für Ereignisse

Da die RPVM die Modellelemente der ausgeführten Sprache und deren Lebenszyklen nicht kennt, ist sie nicht in der Lage, Ereignisse für diese Elemente zu erzeugen. Die Erzeugung der Ereignisse wird daher der jeweiligen Sprache überlassen. Die RPVM sieht entsprechend dem Fabrikentwurfsmuster (Gamma et al. 1994) hierfür die Schnittstelle *EventFactory* vor:

```

interface EventFactory<E> {
    Iterable<? extends Event<E>> createEvents(E element);
}
  
```

Für jeden Elementtyp (Aktivität, Datenobjekt usw.) ist eine Implementierung der Schnittstelle nötig. Sie ist daher mit der Typvariablen E parametrisiert. Erst bei der Implementierung der Schnittstelle wird E durch den konkreten Typ des Elements ersetzt, für das die Implementierung Ereignisse erzeugt. Der Aufruf von `createEvents` erzeugt dann für ein Modellelement `element` die möglichen Ereignisse und gibt sie in Form eines iterierbaren Objekts (`Iterable`) zurück. Dies ist die allgemeinste Form einer Menge von Elementen. Ob es sich um eine Menge oder eine Liste handelt, wird also durch die Schnittstelle nicht festgelegt und kann der jeweiligen Implementierung angepasst werden. Die RPVM ist damit in der Lage sprachspezifische Ereignisse zu erzeugen um sie anschließend bewerten zu können.

5.3 Bewertung von Ereignissen und Instanzen

Nach der Generierung der Ereignisse verfügt die RPVM für eine Prozessinstanz über alle Ereignisse, die in dieser Prozessinstanz auftreten können. Die Generierung der Ereignisse geschieht ohne Berücksichtigung der modellierten Prozessregeln. Der innovative Kern der Ausführung besteht nun darin, diese möglichen Ereignisse auf der Grundlage der Prozessregeln zunächst zu bewerten und anschließend einzuschränken. Jedes der möglichen Ereignisse wird dazu mit dem bisherigen Ereignisverlauf der Instanz kombiniert und für jeden entstehenden Verlauf wird entschieden, inwiefern er den Regeln entspricht. Dabei muss entschieden werden, ob ein simuliertes Ereignis und damit eine mögliche Handlung im Prozess *verboten*, *nicht empfohlen*, *neutral* oder *empfohlen* ist.

Die RPVM stellt selbst nur minimale Anforderungen an ein regelbasiertes Modell. Ein Prozessmodell \mathcal{M} aus Sicht der RPVM setzt sich aus den Modellelementen \mathcal{E} und den Regeln \mathcal{R} zusammen.

$$\mathcal{M} = (\mathcal{E}, \mathcal{R})$$

Dabei sind die Modellelemente nur diejenigen, für die Ereignisse auftreten können, also zum Beispiel Aktivitäten und Datenobjekte. Eine Regel r ist eine logische Aussage, die erfüllt oder verletzt sein kann. Die Menge der Regeln \mathcal{R} eines Prozessmodells lässt sich in *harte Regeln* \mathcal{R}_H , *weiche Regeln* \mathcal{R}_S und *Meilensteine* \mathcal{R}_M unterteilen.

$$\mathcal{R} = \mathcal{R}_H \cup \mathcal{R}_S \cup \mathcal{R}_M$$

Eine *Prozessinstanz* \mathcal{I} ist eine Menge von eingetretenen Ereignissen innerhalb eines Prozesses. Für die RPVM müssen diese Ereignisse keine Ordnung haben. Eine Instanz gilt als abgeschlossen, wenn alle definierten Meilensteine erfüllt sind. Sind keine Meilensteine definiert, kann nicht bestimmt werden, wann der entsprechende Prozess als abgeschlossen gilt. In diesem Fall gilt eine Prozessinstanz für die RPVM nie als beendet und wird erst beim Beenden der PN-Plattform abgebrochen.

Um die folgenden Definitionen syntaktisch zu vereinfachen, wird die *Änderung* δ von einer Menge L zu einer anderen Menge R definiert. Eine Änderung ist ein Tupel aus der Menge der entfernten und der Menge der eingefügten Elemente:

$$\delta(L, R) = (L \setminus R, R \setminus L)$$

Für die Änderung zum Beispiel der Menge $\{a, b\}$ zur Menge $\{b, c\}$ gilt somit:

$$\delta(\{a, b\}, \{b, c\}) = (\{a\}, \{c\})$$

5.3.1 Ereignisbewertung

Eine *Bewertung* ρ ist allgemein eine Menge von verletzten Regeln aus \mathcal{R} . Daher gilt immer:

$$\rho \in \mathcal{R}$$

Die *Ereignisbewertung* $\rho(\mathcal{I} \cup e)$ ist die Bewertung einer Instanz \mathcal{I} nach dem Eintreten eines Ereignisses e . Ein Ereignis gilt als **verboten**, wenn sich die Bewertung nach dessen Eintreten mit der Menge der harten Regeln überschneidet. Entsprechend gilt es als **erlaubt**, wenn diese Schnittmenge leer ist.

$$\begin{aligned} \text{forbidden}(e) &\leftarrow \rho(\mathcal{I} \cup e) \cap \mathcal{R}_H \neq \emptyset \\ \text{allowed}(e) &\leftarrow \rho(\mathcal{I} \cup e) \cap \mathcal{R}_H = \emptyset \end{aligned}$$

Ein Ereignis gilt als **final**, wenn Meilensteine definiert wurden und die Bewertung des Ereignisses sich nicht mit diesen überschneidet. Durch das Eintreten des Ereignisses werden also alle Meilensteine erfüllt.

$$\text{final}(e) \leftarrow \mathcal{R}_M \neq \emptyset \wedge \rho(\mathcal{I} \cup e) \cap \mathcal{R}_M = \emptyset$$

5.3.2 Änderung der Instanzbewertung

Die Instanzbewertungsänderung $\Delta\rho_{\mathcal{I}}$ ist die Änderung der Bewertung einer Instanz \mathcal{I} mit dem Eintreten eines Ereignisses e . Sie drückt somit die Auswirkung des Ereignisses auf die Instanz aus.

$$\Delta\rho_{\mathcal{I}}(e) = \delta(\rho(\mathcal{I}), \rho(\mathcal{I} \cup e))$$

Ein Ereignis e gilt als **neutral**, wenn seine Instanzbewertungsänderung leer ist, es also keine Auswirkung auf die Instanz hat.

$$\text{neutral}(e) \leftarrow \Delta\rho_{\mathcal{I}}(e) = (\emptyset, \emptyset)$$

Ein Ereignis gilt als **empfohlen**, wenn es erlaubt ist und zur Aufhebung von Verstößen gegen weiche Regeln oder Meilensteine führt und keine neuen Regeln verletzt werden.

$$\begin{aligned} \text{recommended}(e) &\leftarrow \Delta\rho_{\mathcal{I}}(e) = (P, \emptyset) \\ &\wedge P \cap (\mathcal{R}_S \cup \mathcal{R}_M) \neq \emptyset \end{aligned}$$

Im Gegenzug gilt es als **nicht empfohlen**, wenn erlaubt ist aber zu Verstößen gegen weiche Regeln oder Meilensteine führt.

$$\begin{aligned} \text{notRecommended}(e) &\leftarrow \Delta\rho_{\mathcal{I}}(e) = (P, Q) \\ &\wedge Q \cap (\mathcal{R}_S \cup \mathcal{R}_M) \neq \emptyset \end{aligned}$$

5.3.3 Änderung der Ereignisbewertung

Die Ereignisbewertungsänderung ΔV_e ist die Änderung der Ereignisbewertung vom vorherigen zum aktuellen Ausführungsschritt:

$$\Delta\rho_e(e) = \delta(\rho(\mathcal{I} \cup e)_{t-1}, \rho(\mathcal{I} \cup e)_t)$$

mit $\rho(\mathcal{I} \cup e)_t = \emptyset$ für $t < 0$

Demnach gilt ein Ereignis als **verändert**, wenn dessen Ereignisbewertungsänderung nicht leer ist oder es sich um den ersten Zeitschritt handelt.

$$\begin{aligned} \text{changed}(e) &\leftarrow \Delta\rho_e(e) \neq (\emptyset, \emptyset) \\ \vee t &= 0 \end{aligned}$$

Ein Ereignis gilt als **neu erlaubt**, wenn es im letzten Zeitschritt verboten war und im aktuellen erlaubt ist. Es gilt ebenfalls als neu erlaubt, wenn es im aktuellen erlaubt ist und es sich um den ersten Zeitschritt handelt, also unmittelbar nach dem Start einer Prozessinstanz. In diesem Fall liegen noch keine vorherigen Ereignisse vor.

$$\begin{aligned} \text{newlyAllowed}(e) &\leftarrow \Delta\rho_e(e) = (P, Q) \\ &\wedge Q \cap \mathcal{R}_H = \emptyset \\ &\wedge (P \cap \mathcal{R}_H \neq \emptyset) \vee t = 0 \end{aligned}$$

Ein Ereignis gilt als **neu verboten**, wenn es im letzten Zeitschritt erlaubt war und im aktuellen verboten ist. Wie oben gilt es ebenfalls als neu verboten, falls es im aktuellen Zeitschritt verboten ist und es sich um den ersten Zeitschritt handelt.

$$\begin{aligned} \text{newlyForbidden}(e) &\leftarrow \Delta\rho_e(e) = (P, Q) \\ &\wedge Q \cap \mathcal{R}_H \neq \emptyset \\ &\wedge (P \cap \mathcal{R}_H = \emptyset) \vee t = 0 \end{aligned}$$

5.4 Beispiel

Um die Funktionsweise der RPVM zu illustrieren, soll im Folgenden die Interpretation einer einfachen exemplarischen Modellierungssprache betrachtet werden. Die Sprache sieht lediglich den Modellelementtyp *Activity* mit die Ereignistypen *Start* und *Complete* vor. Das Prädikat $\text{Complete}(e)$ besagt damit, dass e ein *Complete*-Ereignis ist. Das Prädikat $\text{element}(e, a)$ besagt, dass sich Ereignis e auf Aktivität a bezieht. Als Beispiel soll ein einfaches Modell mit den Aktivitäten A , B und C und den Regeln $\mathcal{R}_H \cup \mathcal{R}_S \cup \mathcal{R}_M$ dienen. Die Regeln enthalten die logischen Operatoren \wedge (und) und \rightarrow (Implikation), wobei $a \rightarrow b \equiv \neg(a \wedge \neg b)$ ist. Außerdem enthalten sie die Quantoren \forall und \exists .

Die harten Regeln h_1 und h_2 fordern, dass eine Aktivität gestartet worden sein muss, bevor sie beendet werden kann. Regel h_2 fordert, dass dem Start von B die Beendigung von A vorausgehen muss. Die weiche Regel s empfiehlt, dass dem Start von C die Beendigung von A vorausgehen sollte und der Meilenstein m ist mit der Beendigung von B erfüllt:

$$\begin{aligned} \mathcal{R}_H = \{ \quad & h_1: \quad \forall a, \exists c, \exists s : \text{Complete}(c) \wedge \text{element}(c, a) \rightarrow \text{Start}(s) \wedge \text{element}(s, a), \\ & h_2: \quad \exists s, \exists c : \text{Start}(s) \wedge \text{element}(s, B) \rightarrow \text{Complete}(c) \wedge \text{element}(c, A) \} \\ \mathcal{R}_S = \{ \quad & s: \quad \exists s, \exists c : \text{Start}(s) \wedge \text{element}(s, C) \rightarrow \text{Complete}(c) \wedge \text{element}(c, A) \} \\ \mathcal{R}_M = \{ \quad & m: \quad \exists c : \text{Complete}(c) \wedge \text{element}(c, B) \} \end{aligned}$$

Tabelle 5.1 Zustand und Bewertungen zum ersten Ausführungsschritt

$t = 0$						
\mathcal{I}	$\rho(\mathcal{I})$	Ereignis	$\rho(\mathcal{I} \cup e)$	$\Delta\rho\mathcal{I}(e)$	Bewertung	
\emptyset	$\{m\}$	Start(A)	$\{m\}$	(\emptyset, \emptyset)	neutral	•
		Complete(A)	$\{m, h_1\}$	$(\emptyset, \{h_1\})$	verboten	•
		Start(B)	$\{m, h_2\}$	$(\emptyset, \{h_2\})$	verboten	•
		Complete(B)	$\{h_1\}$	$(\{m\}, \{h_1\})$	verboten	•
		Start(C)	$\{m, s\}$	$(\emptyset, \{s\})$	nicht empfohlen	•
		Complete(C)	$\{m, h_1\}$	$(\emptyset, \{h_1\})$	verboten	•

Tabelle 5.1 zeigt den Zustand der Instanz zum ersten Ausführungsschritt. Zum Zeitpunkt $t = 0$ sind noch keine Ereignisse eingetreten, also $\mathcal{I} = \emptyset$. Der Meilenstein m wird in diesem Zustand noch nicht erfüllt, die Bewertung der Instanz ist also $\rho(\mathcal{I}) = \{m\}$. In der dritten Spalte „Ereignis“ sind die möglichen Ereignisse für die Instanz aufgezählt. Die vierte Spalte enthält die Bewertung der Instanz nach dem Eintreten des jeweiligen Ereignisses, also $\rho(\mathcal{I} \cup e)$. Die fünfte Spalte enthält die Änderung der Bewertung nach dem Eintreten des Ereignisses $\Delta\rho\mathcal{I}(e)$ gegenüber der bisherigen Bewertung der Instanz $\rho(\mathcal{I})$. Die sechste Spalte enthält die resultierende Bewertung des Ereignisses. Das Starten von A ist als neutral, das von C als nicht empfohlen bewertet. Die siebte Spalte gibt an, ob sich die Bewertung des Ereignisses gegenüber dem letzten Zeitschritt verändert hat. Nur veränderte Bewertungen werden an die Implementierung der ausgeführten konkreten Sprache übergeben. Sie kann diese Informationen nun entsprechend interpretieren und zum Beispiel für die beiden Aktivitäten entsprechende Aufgabeninstanzen in der Aufgabenverwaltung der CPI anlegen und den Prozessteilnehmern zusammen mit der Bewertung anbieten.

Für den zweiten Ausführungsschritt nehmen wir an, dass Aktivität A gestartet wird, das entsprechende Ereignis tritt also ein und wird an den Verlauf angehängt. Die Instanz ist damit $\mathcal{I} = \{\text{Start(A)}\}$ und der zweite Ausführungsschritt wird angestoßen.

Tabelle 5.2 Zustand und Bewertungen zum zweiten Ausführungsschritt

$t = 1$						
\mathcal{I}	$\rho(\mathcal{I})$	Ereignis	$\rho(\mathcal{I} \cup e)$	$\Delta\rho\mathcal{I}(e)$	Bewertung	
$\{\text{Start(A)}\}$	$\{m\}$	Start(A)	$\{m\}$	(\emptyset, \emptyset)	neutral	
		Complete(A)	$\{m\}$	(\emptyset, \emptyset)	neutral	•
		Start(B)	$\{m, h_2\}$	(\emptyset, \emptyset)	verboten	
		Complete(B)	$\{h_1\}$	$(\{m\}, \{h_1\})$	verboten	
		Start(C)	$\{m, s\}$	$(\emptyset, \{s\})$	nicht empfohlen	
		Complete(C)	$\{m, h_1\}$	$(\emptyset, \{h_1\})$	verboten	

Nach dem zweiten Ausführungsschritt in Tabelle 5.2 ergibt sich nur für das Ereignis **Complete(A)** eine Änderung der Bewertung. Das Abschließen der Aktivität wird statt als verboten nun als neutral bewertet. Über die Aufgabenverwaltung könnte nun zum Beispiel direkt das Abschließen

der entsprechenden Aufgabeninstanz erlaubt werden. Das Abschließen der Aktivität stößt die Ausführung erneut an.

Tabelle 5.3 Zustand und Bewertungen zum dritten Ausführungsschritt

$t = 2$						
\mathcal{I}	$\rho(\mathcal{I})$	Ereignis	$\rho(\mathcal{I} \cup e)$	$\Delta\rho\mathcal{I}(e)$	Bewertung	
{ Start(A), Complete(A) }	{m}	Start(A)	{m}	(\emptyset, \emptyset)	neutral	
		Complete(A)	{m}	(\emptyset, \emptyset)	neutral	
		Start(B)	{m}	(\emptyset, \emptyset)	neutral	•
		Complete(B)	{h ₁ }	({m}, {h ₁ })	verboten	
		Start(C)	{m}	(\emptyset, \emptyset)	neutral	•
		Complete(C)	{m, h ₁ }	($\emptyset, \{h_1\}$)	verboten	

Tabelle 5.3 zeigt den Zustand der Instanz zum dritten Zeitschritt. Das Beginnen der Aktivität B verstößt nun nicht weiter gegen die harte Regel h₂, wodurch sich eine Änderung ergibt und das Beginnen von B nun als neutral bewertet wird. Das Beginnen von C verstößt nun nicht mehr gegen die weiche Regel s, wodurch C nicht länger als nicht empfohlen sondern nun als neutral gilt. Im dritten und letzten Zeitschritt in Tabelle 5.4 wird schließlich Aktivität B gestartet und das entsprechende Ereignis tritt ein.

Tabelle 5.4 Zustand und Bewertungen zum vierten Ausführungsschritt

$t = 3$						
\mathcal{I}	$\rho(\mathcal{I})$	Ereignis	$\rho(\mathcal{I} \cup e)$	$\Delta\rho\mathcal{I}(e)$	Bewertung	
{ Start(A), Complete(A), Start(B) }	{m}	Start(A)	{m}	(\emptyset, \emptyset)	neutral	
		Complete(A)	{m, h ₁ }	($\emptyset, \{h_1\}$)	verboten	
		Start(B)	{m}	(\emptyset, \emptyset)	neutral	
		Complete(B)	\emptyset	({m}, \emptyset)	empfohlen	•
		Start(C)	{m}	(\emptyset, \emptyset)	neutral	
		Complete(C)	{m, h ₁ }	($\emptyset, \{h_1\}$)	verboten	

Nach dem Beginnen von B ist nun auch das Abschließen von B erlaubt. Die Instanz würde nach dem Abschließen von B nicht weiter gegen den Meilenstein m verstoßen und es kämen auch keine neuen Verstöße hinzu. Daher gilt das Abschließen von B als empfohlen. Nach dem Eintreten des Ereignisses wären alle definierten Meilensteine erfüllt und die Instanz würde von der RPVM beendet werden.

5.5 Zwei Facetten einer Prozessregel

Die Ausführung der RPVM basiert auf der Simulation des nächsten Ereignisses im Prozessverlauf. Dabei wird nicht nur jeweils das simulierte nächste Ereignis allein bewertet, sondern die Kombination aus ihm und dem bisherigen Verlauf einer Prozessinstanz. Dadurch fließen neben der nächsten auch alle bisherigen Handlungen in einer Prozessinstanz in die Bewertung ein. Eine

Prozessregel muss daher unter Umständen zwei Rollen erfüllen: die Vorwärts- und die Rückwärtsbewertung. Diese Tatsache muss beim Entwurf der Regeln beachtet werden, damit zum Beispiel der Verstoß gegen eine Regel die gesamte Laufzeit einer Instanz über bestehen bleibt. Zur Erklärung soll Regel s aus dem in Abschnitt 5.4 beschriebenen Beispiel dienen:

$$\exists s, \exists c : \text{Start}(s) \wedge \text{element}(s, \mathbf{C}) \rightarrow \text{Complete}(c) \wedge \text{element}(c, \mathbf{A})$$

Die Regel fordert den Abschluss von A vor dem Beginn von C und nutzt dazu eine entsprechende Implikation. Der Zusammenhang lässt sich wie in Tabelle 5.5 gezeigt als Wahrheitstabelle aufstellen.

Tabelle 5.5 Wahrheitstabelle zur exemplarischen Regel

Start(C)	Complete(A)	Start(C) \rightarrow Complete(A)
wahr	wahr	wahr
wahr	falsch	falsch
falsch	wahr	wahr
falsch	falsch	wahr

Die Regel verbietet also schlicht das Beginnen von C ohne den Abschluss von A. Der Verlauf

$$\{ \text{Start}(\mathbf{A}), \text{Start}(\mathbf{C}) \}$$

würde die Regel also verletzen, der Verlauf

$$\{ \text{Start}(\mathbf{A}), \text{Complete}(\mathbf{A}), \text{Start}(\mathbf{C}) \}$$

hingegen würde sie erfüllen. Wichtig ist, dass in diesem Fall der gewünschte Verlauf resultiert, ohne dass die Regel eine zeitliche Komponente enthält. Die zeitliche Ordnung von **Start(C)** und **Complete(A)** wird nicht berücksichtigt. **Start(C)** erfordert die Existenz von **Complete(A)**, das damit zwingend vor **Start(C)** eingetreten sein muss, da es sich bereits im bisherigen Verlauf befindet. Die Vorwärtssimulation erfolgt also allein durch das Ausführungsprinzip auch ohne zeitlichen Bezug.

Problematisch wird der fehlende zeitliche Bezug jedoch während der späteren Ausführung für weiche Regeln, also Empfehlungen. Wurde die Regel missachtet, was für weiche Regeln möglich ist, dann kann zum Beispiel folgender Verlauf entstehen:

$$\{ \text{Start}(\mathbf{A}), \text{Start}(\mathbf{C}), \text{Complete}(\mathbf{C}) \}$$

Aktivität A wurde also gestartet aber nicht abgeschlossen und Aktivität C direkt im Anschluss durchgeführt, obwohl dies Regel s verletzt. Dieser Verlauf widerspricht also momentan Regel s. Würde jetzt Aktivität A abgeschlossen werden, dann würden beide Ereignisse **Start(C)** und **Complete(A)** vorliegen und der Verlauf würde nicht weiter gegen Regel s verstoßen:

$$\{ \text{Start}(\mathbf{A}), \text{Start}(\mathbf{C}), \text{Complete}(\mathbf{C}), \text{Complete}(\mathbf{A}) \}$$

Der frühere Verstoß gegen die Empfehlung geht also verloren. Wenn diese rückwärtige Bewertung erhalten bleiben soll, müssen die Zeitstempel der Ereignisse betrachtet werden. Die Regel muss enthalten, dass das Beginnen von C den *vorherigen* Abschluss von A erfordert:

$$\exists s, \exists c : \text{Start}(s) \wedge \text{element}(s, \mathbf{C}) \wedge \text{time}(s, t_s) \rightarrow \text{Complete}(c) \wedge \text{element}(c, \mathbf{A}) \wedge \text{time}(c, t_c) \\ \wedge t_c < t_s$$

In diesem Fall ließe sich ein Verstoß gegen diese Regel nicht durch weitere Ereignisse umkehren. Das Beispiel zeigt, dass oft schlankere Regeln genügen würden, um nur die Vorwärtsbewertung abzudecken. Bei weichen Regeln ist es allerdings notwendig auch die Rückwärtsbewertung zu beachten.

5.6 Auswertung von Regeln

Bisher wurde gezeigt, wie die RPVM Ereignisse erzeugt und Ereignisverläufe bewertet. Zur Bewertung von Ereignisverläufen muss ermittelt werden, welchen Regeln der Verlauf entspricht und welche er verletzt. Die für einen Prozess festgelegten Regeln müssen also auf der Basis der Modellelemente und der bisherigen Ereignisse effizient ausgewertet werden. Der Rete-Algorithmus erfüllt diese Aufgabe und ist weit verbreitet, so dass auf zuverlässige und leistungsfähige Implementierungen zurückgegriffen werden kann.

5.6.1 Rete-Algorithmus

Der Rete-Algorithmus ist ein Mustererkennungsverfahren, das für eine Menge von Mustern diejenigen bestimmt, die durch eine Menge an Objekten erfüllt werden (Forgy 1982). Damit eignet sich der Algorithmus zur Auswertung der oben genannten Regeln (Muster) auf der Basis der Modellelemente und Ereignisse (Objekte).

Der Algorithmus erzeugt aus den Mustern ein Netzwerk aus α -Knoten, α -Speichern, β -Knoten und β -Speichern. α -Knoten sind Selektionsbedingungen auf einzelnen Objekten. α -Speicher enthalten Referenzen auf die Objekte, die einem Muster entsprechen. β -Knoten verknüpfen mehrere einzelne Bedingungen, wobei β -Speicher das Ergebnis dieser Verknüpfung enthalten.

Als Beispiel soll das Muster $\text{weiblich}(x) \wedge \text{kennt}(x, y) \wedge \text{männlich}(y)$ dienen, also alle Bekanntschaften einer weiblichen zu einer männlichen Person. Die Symbole **weiblich**, **kennt** und **männlich** sind logische Prädikate; x und y sind Variablen. Der Rete-Algorithmus baut für die in Abbildung 5.3 aufgezählten Fakten (1) das entsprechende Rete-Netzwerk auf (2). Die genaue Vorgehensweise ist implementierungsspezifisch.

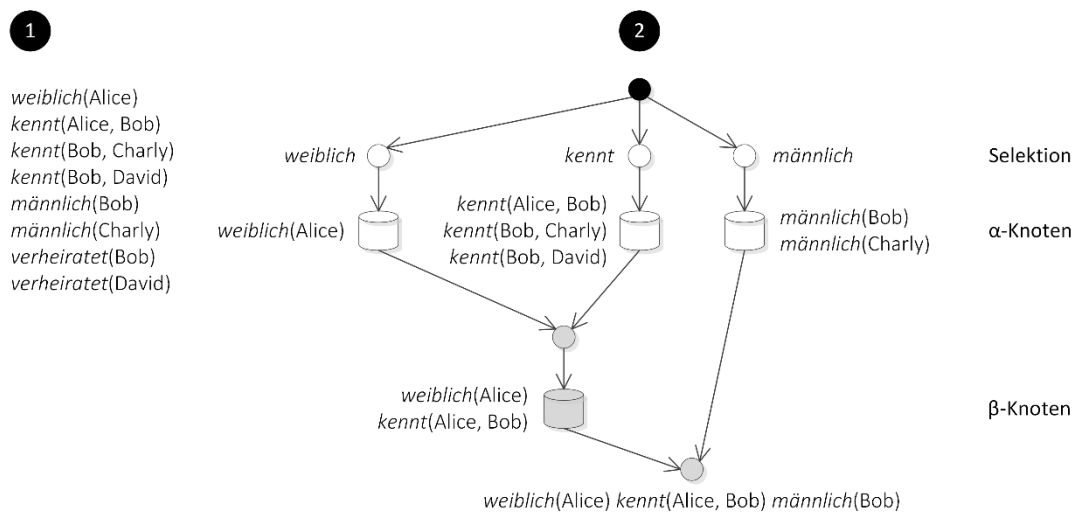


Abbildung 5.3 Beispiel für ein Rete-Netzwerk

Für die drei Einzelbedingungen werden drei α -Knoten angelegt und für die Verknüpfungen zwei β -Knoten. Das Ergebnis des letzten Knotens bildet die Teilmenge der Objekte, die diese Regel erfüllen. Der Algorithmus liefert also für die gegebene Objektmenge diejenigen, die dem Muster entsprechen.

Der eigentliche Gewinn des Verfahrens macht sich erst für mehrere Regeln bemerkbar. In diesem Fall wird nicht für jede Regel ein eigenes Netz aufgebaut, sondern es werden gleiche Teile der einzelnen Netze verschmolzen. Um alle Regeln auszuwerten, kann dann das resultierende Rete-Netzwerk ausgewertet werden, statt jede Regel einzeln zu prüfen. Der Rete-Algorithmus stellt damit eine effiziente Grundlage für die Auswertung von Prozessregeln dar und soll daher in der RPVM zum Einsatz kommen.

5.6.2 Drools und DRL

Die Drools-Plattform beinhaltet eine aktuelle, zuverlässige und leistungsfähige Implementierung des oben gezeigten Rete-Verfahrens (JBoss Drools Team 2013). Zudem steht sie als Bibliothek frei zur Verfügung und soll daher in die RPVM integriert werden. Drools umfasst die Sprache *Drools Rule Language* (DRL), mit der sich Regeln formulieren lassen. Über entsprechende Schnittstellen lassen sich dann Objektmengen laden und Regeln auf diesen Objekten auswerten. Der Sprachumfang von DRL soll im Folgenden kurz beschrieben werden.

Regeln in DRL bestehen aus einer Bedingung und einer Konsequenz. Trifft die Bedingung zu, wird die Konsequenz ausgeführt. Als Bedingung einer Regel unterstützt die DRL den Sprachumfang der Prädikatenlogik erster Stufe. Regeln können durch beliebige Metadatenattribute markiert werden:

```

@key1(value1)
...
@keyn(valuen)
rule Rule1
when ...
then ...
end

```

Um Teile von Bedingungen zu benennen und wiederzuverwenden, können Anfragen (engl. *query*) definiert werden. Eine Anfrage kann Parameter entgegennehmen und enthält als Rumpf wiederum eine Bedingung:

```

query peopleFromOlderThan(int minAge, String from)
    $person : Person(age > minAge, location == from)
end

```

Bedingungen sind aus Mustern zusammengesetzt. Ein Muster (engl. *pattern*) für Objekte besteht aus dem Typ der Objekte und Einschränkungen (engl. *constraints*) ihrer Eigenschaften. Die zu einem Muster passenden Objekte lassen sich bei jeder Übereinstimmung an eine Variable binden und somit innerhalb der Bedingung wiederverwenden.

Folgendes Muster passt zu allen weiblichen Kunden, die älter als 35 Jahre sind, und bindet die entsprechenden Objekte jeweils an die Variable *c*:

```
$c : Customer(sex == FEMALE, age > 35)
```

Auch die Eigenschaften der Objekte lassen sich an eine Variable binden. So wird im Folgenden das Alter aller männlichen Kunden jeweils an die Variable *\$a* gebunden:

```
Customer(sex == MALE, $a : age)
```

Die DRL unterstützt auch den Zugriff auf verschachtelte Eigenschaften und Methoden:

```

Person(address.houseNumber == 50)
Person(getAddress().getHouseNumber() == 50)

```

Die logische Konjunktion zweier Muster wird mit **and** gebildet. Folgender Ausdruck liefert alle Kombinationen aus Kunden und Lieferanten, deren Ort übereinstimmt:

```
Customer($c : city) and Supplier(city == $c)
```

Die logische Disjunktion wird mit **or** gebildet. Regeln, die eine Disjunktion enthalten, werden in eine disjunkte Normalform transformiert, so dass effektiv mehrere unabhängige Regeln entstehen. Aus diesem Grund ist der Geltungsbereich gebundener Variablen unterhalb einer Disjunktion auf den jeweiligen Ast der Disjunktion beschränkt. Die Disjunktion erlaubt dadurch eine wahlweise Bindung von Variablen. So wird im folgenden Beispiel die Variable *\$pensioner* an die Vereinigung der beiden Objektmenge gebunden:

```
$pensioner : (Person(sex == FEMALE, age > 60) or Person(sex == MALE, age > 65))
```

Der Existenzquantor wird mit **exists** gebildet. Der entsprechende Ausdruck wird höchstens einmal ausgewertet und es werden keine Variablen gebunden. Folgende Regel fordert, dass mindestens eine Frau existiert, die älter als 35 Jahre ist:

```
exists(Customer(sex == FEMALE, age > 35))
```

Der Allquantor wird mit **forall** gebildet. Der Ausdruck ist wahr, wenn alle Fakten, die auf das erste Muster passen, auch auf alle weiteren Muster passen. Variablen, die innerhalb des Allquantors gebunden werden, sind außerhalb von ihm nicht sichtbar. Folgende Regel fordert, dass alle englischen Busse rot sind:

```
forall($bus : Bus(type == 'english')
        Bus(this == $bus, color = 'red'))
```

Die Negation wird schließlich mit **not** gebildet:

```
not(Bus(color == "red"))
```

5.6.3 Integration von Drools in die RPVM

Die Drools-Plattform soll genutzt werden, um Prozessregeln auszuwerten und wird daher in die RPVM integriert. Als Beispiel für Verknüpfung von Drools mit der RPVM soll erneut die fiktive Prozessmodellierungssprache aus Abschnitt 5.4 dienen, die nur Modellelement vom Typ *Activity* mit den Ereignissen *Start* und *Complete* enthält. Ein entsprechendes Prozessmodell sei $\mathcal{M} = (\mathcal{E}, \mathcal{R})$ mit $\mathcal{E} = \{A, B\}$ mit der Regel $\mathcal{R} = \{(S, "A \text{ vor } B", \text{Start}(s) \wedge \text{element}(s, B) \rightarrow \text{Complete}(c) \wedge \text{element}(c, A))\}$. Es enthält also zwei Aktivitäten A und B und eine weiche Regel, die besagt, dass der Start von B das Abschließen von A voraussetzt. Vor dem ersten Auswertungsschritt der Prozessinstanz wird ein Arbeitsspeicher angelegt, der die Fakten, also die Modellelemente und den Ereignisverlauf enthält. Im Beispiel wird also ein Arbeitsspeicher einem zunächst leeren Verlauf aus *Start* und *Complete*-Ereignissen angelegt. Die Modellelemente werden später zusammen mit den Regeln geladen. Der Arbeitsspeicher wird während der gesamten Laufzeit der Prozessinstanz wiederverwendet.

Um die Drools-Plattform zur Bewertung, also zur Auswertung der Prozessregeln nutzen zu können, müssen die Prozessregeln in der DRL-Sprache vorliegen. Dabei bildet nur die Bedingung (**when**) die eigentliche Regel, in der Konsequenz (**then**) wird sie lediglich als erfüllt an die RPVM gemeldet. Die Variable `listener` wird zuvor an ein Objekt gebunden, das die Schnittstelle `RuleListener` implementiert:

```
interface RuleListener {
    void onRuleOccured(org.drools.core.rule.Rule rule);
}
```

Die Schnittstelle sieht eine Methode `onRuleOccured` vor, die aufgerufen wird, sobald die übergebene Regel ausgelöst wird. Metadatenattribute wie `@type("soft")` werden genutzt, um eine Regel als harte Regel, weiche Regel oder Meilenstein zu kennzeichnen und eine Meldung zu hinterlegen. Auch die Modellelemente, also die *Activity*-Instanzen A und B müssen mittels Regeln

angelegt werden. Die entsprechenden Regeln werden durch ein weiteres Metadatenattribut `@type("model")` gekennzeichnet. Das Beispielmmodell entspricht also folgender DRL-Datei:

```
package org.example

import org.example.model.Activity
import org.example.events.Start
import org.example.events.Complete

global de.ubt.ai4.pn.rpvm.RuleListener listener

@type("model")
rule M1
when
then
    insert(new Activity("A", $p));
    insert(new Activity("B", $p));
end

@type("soft")
@message("A vor B")
rule S1
when
    $a: Activity(id == "A")
    and $b: Activity(id == "B")
    and not(Start(element == $b) and not Complete(element == $a))
then
    listener.onRuleOccured(drools.getRule());
end
```

Da DRL keine Implikation unterstützt, wurde die Regel entsprechend der Identität

$$\alpha \rightarrow \beta \equiv \neg(\alpha \wedge \neg\beta)$$

umgeschrieben. Die Strukturen für den Rete-Algorithmus werden nur einmal zum Zeitpunkt der Erzeugung des Ausführungszeigers für eine Prozessinstanz aufgebaut und von da an über die gesamte Dauer der Prozessinstanz hinweg wiederverwendet.

5.7 Ereignisbehandlung

Die Ereignisse, deren Bewertung sich geändert hat, werden zusammen mit dieser Änderung und der Änderung der Instanzbewertung über eine Schnittstelle an die Implementierung der konkreten Modellierungssprache gemeldet. Auf dieser Ebene können dann die Ereignisse, deren Bewertung sich geändert hat, entsprechend behandelt und in Aktionen übersetzt werden. Die Implementierung der Modellierungssprache muss die Schnittstelle `EventHandler` und damit die Methode `onRatingsChanged` implementieren:

```
interface EventHandler<E> {  
    void onRatingsChanged(DeclarativeExecution execution,  
        E element,  
        Class<? extends Event<E>> type,  
        List<RatedEvent<E>> ratedEvents);  
}
```

Die Methode wird von der RPVM mit den folgenden Parametern aufgerufen:

- **execution**: die aktuelle Instanz des Ausführungszeigers
- **element**: das Modellelement, auf das sich alle übergebenen Ereignisse beziehen
- **type**: der Typ der übergebenen Ereignisse
- **ratedEvents**: eine Liste der Ereignisse, deren Bewertung sich gegenüber dem letzten Ausführungsschritt geändert hat

Jedes Ereignis wird dabei in ein **RatedEvent**-Objekt verpackt, sodass die Änderungen der Instanz- und der Ereignisbewertung transportiert werden können. Für jeden Elementtyp (Aktivität, Datenobjekt usw.) muss eine Implementierung der Schnittstelle vorliegen. Sie ist daher mit der Typvariablen **E** parametrisiert, um bei ihrer Implementierung den Typ des Elements festlegen zu können, auf die sich die behandelten Ereignisse beziehen.

5.8 Mehrfach instanziierbare Elemente

Modellelemente können mehrfach instanziiierbar sein, wenn zum Beispiel zu einer modellierten Aktivität mehrere Instanzen parallel zueinander gestartet werden können. Dies bedeutet, dass zu einer Aktivität mehrere startende Ereignisse zunächst ohne die entsprechenden beendenden Ereignisse eintreten. Analog dazu können für dasselbe Datenobjekt mehrere manipulierende Ereignisse auftreten. Die Art des Datenobjekts legt dann fest ob der letzte Wert überschrieben wird oder ob die Werte kombiniert werden sollen. Auch hier wird das Datenobjekt mehrfach instanziiert.

Die Verwaltung der Instanzen von Modellelementen liegt nicht im Aufgabenbereich der RPVM. Sie hat keine Kenntnis von Instanzen und schließt sie daher auch nicht aus. Die Spezialisierung der RPVM für eine konkrete Modellierungssprache muss mehrmalige Ereignisse für dieselben Modellelemente entsprechend interpretieren und die Instanzen und ihre Lebenszyklen entsprechend verwalten.

5.9 Resultierende Architektur der RPVM

Aus den oben beschriebenen Aufgaben der Erzeugung, Bewertung und Behandlung von Ereignissen ergibt sich der in Abbildung 5.4 gezeigte Aufbau für die RPVM. Die RPVM greift auf die Konzepte der CPI und integriert das Drools-Framework.

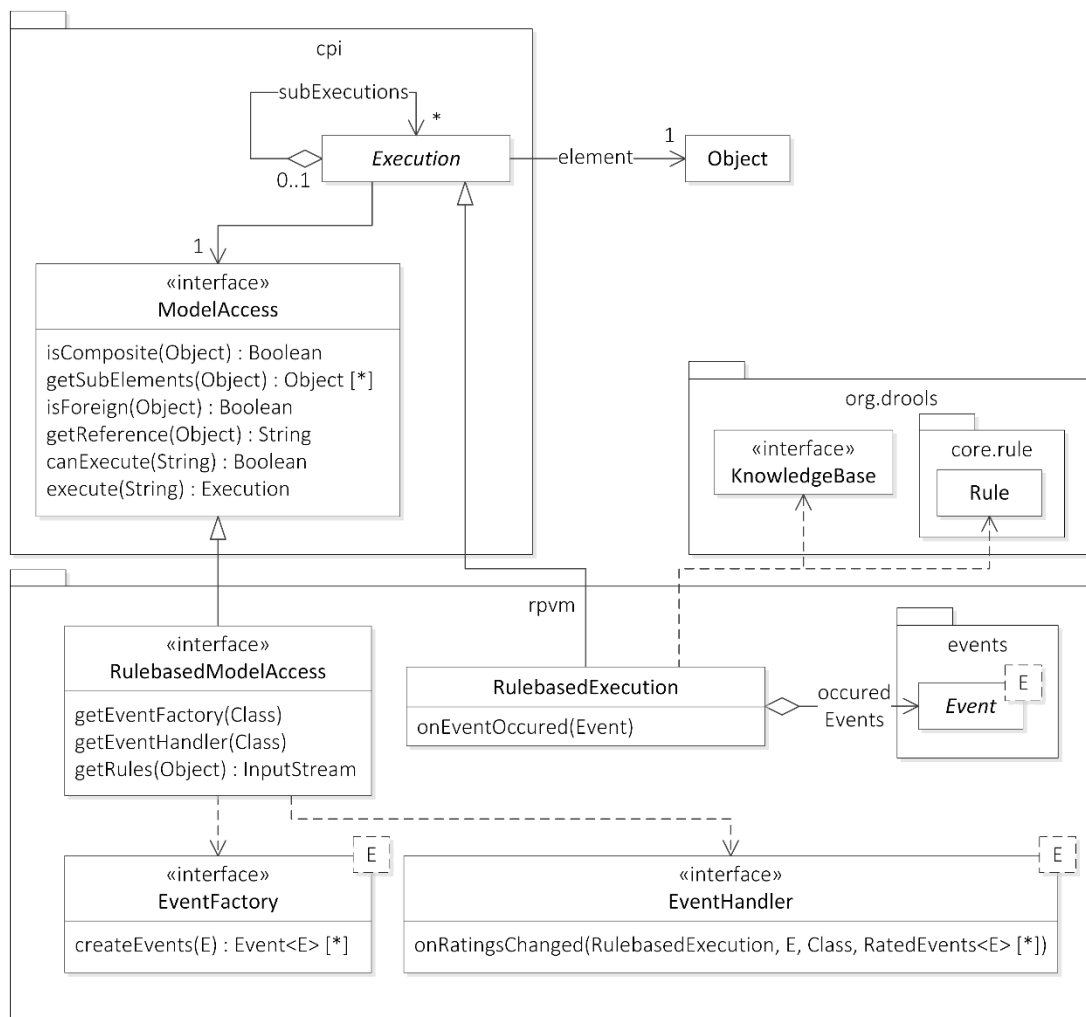


Abbildung 5.4 Aufbau der RPVM

Zentral ist der Ausführungszeiger **RulebasedExecution** für regelbasierte Modelle, der den Ausführungszeiger **Execution** der CPI erweitert. Er deklariert die Methode **onEventOccured**, die die Ausführung nach dem Eintreten eines Ereignisses fortführt. Jeder regelbasierte Zeiger verwaltet den Ereignisverlauf (engl. *log*) für die Elemente, die dem ausgeführten Element untergeordnet sind, in **occuredEvents**. Die Ereignisse müssen die Basisklasse **Event** erweitern. (siehe Abschnitt 5.2). Für die Bewertung von Ereignisverläufen greift die **RulebasedExecution** auf die Klassen **KnowledgeBase** und **Rule** des Drools-Frameworks (**org.drools**) zu. Der sprachspezifische Modellzugriff **ModelAccess** wird für regelbasierte Sprachen durch den **RulebasedModelAccess** erweitert. Er bietet die Möglichkeit, für die verschiedenen Modellelemente einer Sprache Ereignisfabriken (**EventFactory**) und –handler (**EventHandler**) zu verwalten. Die Ereignisfabrik erzeugt die für einen bestimmten Modellelementtyp die möglichen Ereignisse, also zum Beispiel das Zuweisen, Starten und Abschließen für eine Aufgabe oder das Lesen und Schreiben für ein Datenobjekt. Der Ereignishandler reagiert auf geänderte Bewertungen von Ereignissen und löst die

Prozessregeln mit Hilfe des Drools Frameworks bewertet. Falls sich die Bewertung eines möglichen Ereignisses gegenüber dem letzten Ausführungsschritt verändert wird dies an die Implementierung der Sprache gemeldet und kann dort interpretiert und in entsprechende Aktionen übersetzt werden.

Der Gewinn der beschriebenen Architektur liegt in ihrer starken Modularisierung. Statt einem PAS zur Ausführung einer konkreten Modellierungssprache wurde erstmals ein PAS potenziell für eine Vielzahl sowohl an prozeduralen als auch an regelbasierten Modellierungssprachen entworfen.

6 Deklarative Prozesszwischensprache (DPIL)

Zur Modellierung agiler Prozesse eignet sich wie oben gezeigt eine regelbasierte Sprache. Wie in Abschnitt 1.3 beschrieben muss eine solche Sprache alle üblichen Perspektiven abdecken, eine perspektivenübergreifende Modellierung ermöglichen und eine Unterscheidung von Modalitäten erlauben. Wie in Abschnitt 2.2 festgestellt, kann keine der bestehenden regelbasierten Sprachen diese Anforderungen erfüllen. Daher soll im Zuge dieser Arbeit eine entsprechende Sprache entwickelt werden, die sich auf der Basis der oben beschriebenen RPVM ausführen lässt. Da eine Endbenutzerorientierung nicht Teil dieser Arbeit ist, wird die entwickelte Sprache als Zwischensprache verstanden und im Folgenden als Deklarative Prozesszwischensprache (engl. *Declarative Process Intermediate Language*, kurz DPIL) bezeichnet. Grafische Notationen regelbasierter Prozessmodelle müssen zur Ausführung auf die Sprache DPIL abgebildet werden.

Die Sprache DPIL ist eine Programmiersprache, die einem Spezialisten dazu dienen soll, regelbasierte Prozessmodelle in maschineninterpretierbarer Form zu beschreiben. DPIL ist keine Universalsprache wie z.B. Java oder C++, mit der sich ganze Software-Systeme realisieren. Der Sprachumfang ist gerade so gewählt, dass er genau zum Zweck der Prozessmodellierung dient, wodurch sich DPIL leichter erlernen und verwenden lässt als eine Universalsprache. DPIL ist damit eine sogenannte domänenspezifische Sprache (Fowler 2011).

6.1 Abstrakte Syntax der Modellelemente

Die abstrakte Syntax einer Sprache beschreibt ihr Vokabular an Konzepten und wie sie kombiniert werden können um Modelle zu erzeugen (Clark et al. 2008). Die Konzepte der Sprache DPIL werden in strukturelle Elemente und Regeln unterschieden. Strukturelle Modellelemente wie Aktivitäten und Datenobjekte sind nicht spezifisch für das regelbasierte Prinzip. Sie können aus etablierten prozeduralen Sprachen übernommen werden. Dementsprechend sieht DPIL wie auch BPMN Aktivitäten und Datenobjekte als zentrale Elemente eines Prozessmodells vor (Object Management Group Inc. 2011).

6.1.1 Aktivitäten und Datenobjekte

Das Metamodell der strukturellen Elemente von DPIL ist in Abbildung 6.1 dargestellt. Eine Aktivität (**Activity**) stellt einen Arbeitsschritt dar. Ein Prozess (**Process**) ist eine zusammengesetzte Aktivität, eine die also wiederum Aktivitäten enthält. Alle weiteren Arten von Aktivitäten sind elementar. Der entsprechende Arbeitsschritt soll also nicht weiter detailliert werden. Eine personelle Aufgabe (**Task**) wird während des Prozesses einem oder mehreren menschlichen Prozessteilnehmern zugewiesen, die während ihrer Arbeit durch das System unterstützt werden. Eine automatische Operation (**Operation**) ist Arbeit, die durch das System ohne Beteiligung von Menschen erledigt wird. Hierbei kann es sich um den Aufruf eines lokalen Programms oder eines externen Dienstes handeln. Ein Verweis auf einen fremden Prozess (**ProcessReference**) ermöglicht die Einbettung von Subprozessen in anderen Modellierungssprachen. Die genannten Elemente decken die funktionale Perspektive der Modellierung ab (Curtis 1992) und werden in ähnlicher Form auch in BPMN definiert (Object Management Group Inc. 2011).

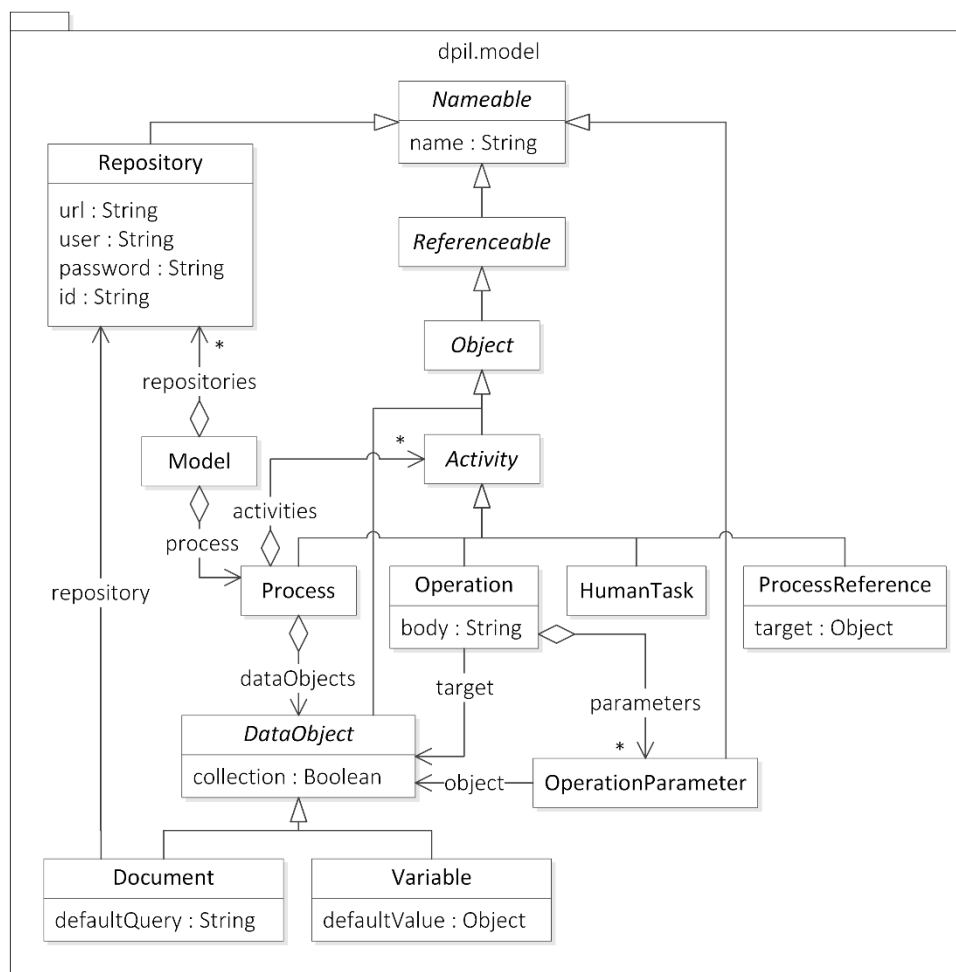


Abbildung 6.1 Modellelemente von DPIL

Außerdem kann ein Prozess Datenobjekte (**DataObject**) definieren. Die können mehrwertig sein (**collection**), also eine Menge von Werten statt nur einen einzelnen repräsentieren. Anders als in BPMN sollen in DPIL zwei Arten von Datenobjekten explizit unterschieden werden. Ein Datenobjekt kann entweder ein Dokument (**Document**) oder eine Variable (**Variable**) sein.

Ein Dokument kapselt einen Verweis auf ein Dokumentenobjekt, das außerhalb des PAS von einem ECM-System verwaltet wird. Das **Repository** kapselt die Verbindungsinformationen für dieses System. Der Lebenszyklus eines Dokuments ist damit von dem eines Prozesses entkoppelt. Dokumente können also in mehreren Prozessen verwendet werden und bestehen weiter auch nachdem Prozesse beendet sind.

Eine Variable kapselt ein beliebiges Objekt, das durch das PAS verwaltet wird. Der Lebenszyklus einer Variablen ist an den des Prozesses gekoppelt. Dadurch können Variablen nur innerhalb ihres Prozesses verwendet werden und verlieren mit Beendigung des Prozesses ihre Gültigkeit. Eine Variable wird technisch als ein Verweis auf ein Objekt im Adressraum des PAS realisiert. Verweist sie auf eine Zeichenkette oder eine Zahl, stellt sie tatsächlich einen flüchtigen Wert dar, der nach Beendigung des Prozesses freigegeben wird. Sie kann jedoch auch auf ein Objekt verweisen, das

mittels objektrelationaler Abbildung mit einem Tupel einer relationalen Datenbank verknüpft ist (Fowler 2002). In diesem Fall wird auf das Objekt und damit auf das Tupel nur während des Prozesses zugegriffen, es kann allerdings bereits vor Beginn des Prozesses und auch nach dessen Beendigung existieren.

Datenobjekte können einen Vorgabewert definieren, also einen Wert, der zu Beginn des Prozesses bereits mit dem Datenobjekt verknüpft wird. Hierzu wird für ein Dokument eine entsprechende Anfrage hinterlegt (`defaultQuery`), die der CMIS-Spezifikation (OASIS 2012) entsprechen muss. Für einfache Dokumente muss diese Anfrage genau ein Objekt zum Ergebnis haben. Für Sammlungen von Dokumenten kann sie mehrere Dokumente liefern, die dann alle als Vorgabewert für das Datenobjekt bereitgestellt werden. Variablen können über `defaultValue` auf ein Objekt oder eine Sammlung von Objekten im Adressraum des Systems verweisen.

Datenobjekte können als Parameter (`OperationParameter`) an eine Operation übergeben werden und als Ziel für ihren Rückgabewert dienen (`target`).

6.2 Ereignisse und Lebenszyklen der Modellelemente

Bisher wurden die strukturellen Bestandteile der Sprache beschrieben, also vor allem Aktivitäten und Datenobjekte. Der entscheidende Bestandteil regelbasierter Prozessmodelle sind Prozessregeln, die die Ereignisse einschränken, die im Lebenszyklus von Aktivitäten und Datenobjekten auftreten können. Zusätzlich zu den strukturellen Elementen sind daher auch Ereignisse Teil der Sprachdefinition. Ihr Metamodell ist in Abbildung 6.2 dargestellt. DPIL erweitert den Basisereignistyp (`Event`) der RPVM. Alle Ereignisse, die sich auf einen Prozessteilnehmer beziehen oder von einem ausgelöst werden, sind personelle Ereignisse (`IdentityEvent`) und verweisen auf eine durchführende Person (`performer`). DPIL greift hierbei auf die Schnittstelle zur Organisationsverwaltung der CPI zurück. In welcher Reihenfolge die Ereignisse auftreten und welche Bedeutung sie im Einzelnen haben, wird im Folgenden beschrieben.

6.2.1 Lebenszyklus von Aufgaben

Personelle Aufgaben können aus Sicht der RPVM begonnen (**Start**) und abgeschlossen (**Complete**) werden. Nur diese beiden Ereignisse werden in den Ereignisverlauf aufgenommen und können daher in Prozessregeln verwendet werden. Innerhalb der Aufgabenverwaltung werden darüber hinaus weitere Zustände für die Reservierung und Unterbrechung von Aufgaben abgebildet. Diese Ereignisse sind aber vor der RPVM verborgen und können daher nicht in Prozessregeln verwendet werden. Der effektive Lebenszyklus ist in Abbildung 6.3 dargestellt. Sobald ein **Start**-Ereignis erlaubt ist, wird eine Aufgabeninstanz in der Aufgabenverwaltung angelegt. Der Lebenszyklus der Instanz beginnt an diesem Punkt. Sind mehrere **Start**-Ereignisse für dieselbe Aufgabe möglich, dann verweisen diese Ereignisse auf verschiedene mögliche Bearbeiter (**performer**). Die Aufgabe wird dann allen möglichen Bearbeitern angeboten. Gibt es nur einen möglichen Bearbeiter, also nur ein mögliches **Start**-Ereignis, wird die Aufgabe direkt für diesen reserviert. Beginnt ein möglicher Bearbeiter die Aufgabe, wird das **Start**-Ereignis ausgelöst und der RPVM gemeldet. Erst mit diesem Schritt hat sich die Prozessinstanz aus Sicht der RPVM verändert. Die Aufgabe ist jetzt in Bearbeitung und kann beliebig unterbrochen und fortgeführt werden. Erst wenn die RPVM das **Complete**-Ereignis für die Aufgabe als erlaubt meldet, gilt die Aufgabe als abschließbar. Auch in diesem Zustand kann sie noch unterbrochen und fortgeführt werden. Wird die Aufgabe durch den Bearbeiter abgeschlossen, wird das **Complete**-Ereignis ausgelöst und wiederum an die RPVM gemeldet. Die Prozessinstanz hat sich erneut verändert und die RPVM führt die Ausführung fort.

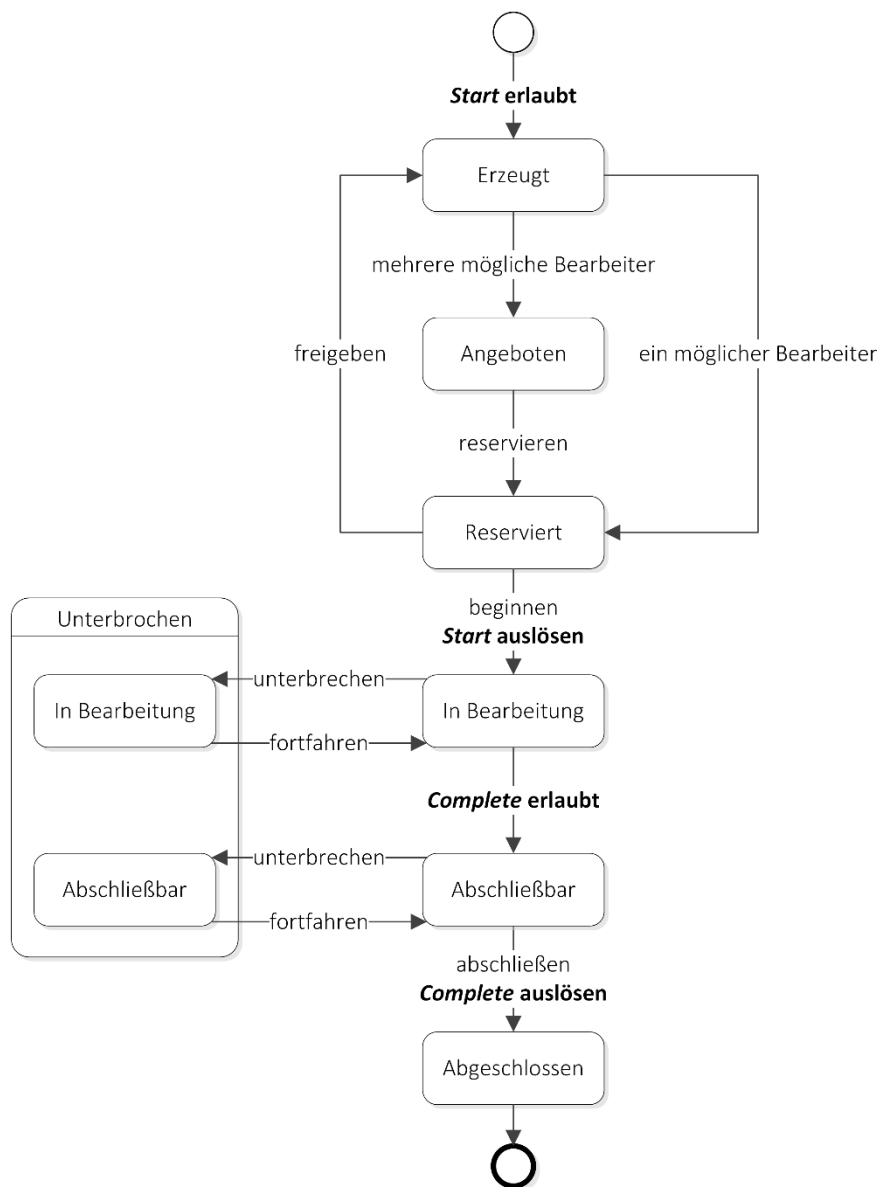


Abbildung 6.3 Lebenszyklus einer Aufgabeninstanz

6.2.2 Lebenszyklus von Datenobjekten

Der Zugriff auf Datenobjekte wird verallgemeinert und auf Lese- (**Read**) und Schreiboperationen (**Write**) abgebildet. Ein tatsächlicher Lesezugriff kann durch das System nicht zuverlässig festgestellt werden. Lesezugriffe werden daher von der RPVM nur als mögliches Ereignis angeboten und entsprechend interpretiert. Sie treten nie ein und werden somit auch nicht an den Ereignisverlauf angehängt. Ein Schreibzugriff hingegen erzeugt einen neuen Wert für das jeweilige Datenobjekt. Für eine Variable wird also ein neuer Verweis auf ein Objekt im Adressraum des Systems hinterlegt (**VariableWrite**) und für ein Dokument eine neuer Verweis auf ein extern verwaltetes Dokument in Form einer CMIS-Objektkennung (**DocumentWrite**). Sobald das Leseereignis für ein Datenobjekt erlaubt ist, wird die Instanz des Datenobjekts in der Aufgabenverwaltung erzeugt und falls angegeben, der Vorgabewert für das Datenobjekt

bereitgestellt. Für Dokumente bedeutet das, dass die entsprechende Anfrage (`defaultQuery`) ausgeführt und die Objektkennung des gefundenen Dokuments als Wert des Datenobjekts hinterlegt wird.

6.2.3 Lebenszyklus von Operationen

Operationen von Diensten können aufgerufen werden (**Invoke**) und kehren anschließend zurück (**Return**). Sobald der Aufruf einer Operation erlaubt ist, wird er durchgeführt und das **Invoke**-Ereignis ausgelöst. Ohne einschränkende Prozessregeln werden also immer alle definierten Operationen beim Start der Prozessinstanz umgehend aufgerufen. Dem Aufruf einer Operation können Datenobjekten als Parameter übergeben werden. Die Operation wird dann mit den aktuellen Werten der Datenobjekte aufgerufen. Wurde der Aufruf durchgeführt, so wird das **Return**-Ereignis ausgelöst. Ein Aufruf kann als Rückgabewert wiederum auf ein Datenobjekt verweisen. In diesem Fall wird für das Datenobjekt zusätzlich zum **Return**-Ereignis des Aufrufs ein **Write**-Ereignis mit dem entsprechenden Wert ausgelöst. Ohne einschränkende Prozessregeln wird eine Operation in jedem Ausführungsschritt der Instanz erneut aufgerufen.

6.3 Spezialisierte abstrakte Syntax für Regeln

Als regelbasierte Modellierungssprache benötigt DPIL Elemente zur Formulierung von Prozessregeln. Wie oben bereits beschrieben, wird zur Auswertung von Prozessregeln die Drools-Plattform in die RPVM integriert. Die Regelsprache der Plattform DRL eignet sich aber nur bedingt zur Definition von Struktur und Regeln eines DPIL-Prozesses. Die strukturellen Elemente des Prozesses müssen in Form von Regeln definiert werden, was vor allem hinsichtlich der verschachtelten Struktur von Aktivitäten sehr viel Code erfordert und fehleranfällig ist. Um zum Beispiel einen Prozess aus zwei Aufgaben zu definieren, ist folgender Abschnitt nötig:

```
@type("model")
rule M1
when
then
    insert(new Process("P"));
end

@type("model")
rule M2
when
    $p: Process(id == "P")
then
    insert(new HumanTask("A", $p));
    insert(new HumanTask("B", $p));
end
```

Dazu kommt, dass die Repräsentation in der Sprache DRL sehr viele wiederkehrende Abschnitte enthält. Jede Prozessregel enthält zum Beispiel dieselbe Konsequenz um die Erfüllung einer Regel an die RPVM zu melden:

```
...  
then  
    listener.onRuleOccured(drools.getRule());  
end
```

Außerdem werden stets dieselben Klassen des DPIL-Metamodells mit denselben Attributen verwendet. Das Importieren ihrer Namen im Kopf des DRL-Artefakts ist damit redundant und fehleranfällig.

Schließlich lässt sich der DRL-Code lediglich auf korrekte DRL-Syntax, nicht aber hinsichtlich der Verwendung in der RPVM validieren. Die Bedeutung der prozessbezogenen Elemente ist der Drools-Plattform nicht bekannt. Es lässt sich also nur sicherstellen, dass der Code gültige DRL-Regeln enthält aber nicht, dass er ein gültiges Prozessmodell darstellt.

Aus diesen Gründen wird eine spezialisierte und exakt festgelegte Syntax für DPIL-Modelle entworfen. Strukturelle Elemente und Regeln sollen sich im Kontext zueinander definieren lassen. Da die Sprache ausschließlich für die Definition von Prozessmodellen und damit für die Verwendung in der RPVM entwickelt wird, kann sie auch entsprechend validiert werden. Da sie zur Formulierung von Regeln dient, eignet sich eine textuelle Syntax (Goldschmidt et al. 2008).

Die abstrakte Syntax der strukturellen Elemente wurde bereits oben definiert. Im Folgenden wird nun eine abstrakte Syntax für Regeln definiert, die sich auf DRL abbilden lässt. Diese bildet also einen weiteren Teil des Metamodells von DPIL und ist in Abbildung 6.4 dargestellt.

Ein DPIL-Modell (`Model`) kann globale Regeln (`GlobalRule`) definieren, also solche, die für alle Prozesse (Ebenen) des Modells gelten. Dabei kann es sich um harte oder weiche Regeln handeln. Für einen Prozess können dann Regeln (`ProcessRule`) angegeben werden, die für die dem Prozess direkt untergeordneten Aktivitäten gelten. Dabei kann es sich neben harten und weichen Regeln auch um Meilensteine handeln. Um bestimmte Regelausdrücke benennen und wiederzuverwenden zu können, können in einem Modell Makros (`Macro`) definiert werden. Regeln und Makros bilden die Klasse der Prädikate (`Predicate`) und enthalten immer einen Regelausdruck (`Expression`). Ein Ausdruck kann binär (`Binary`) sein, also eine Konjunktion (`And`), Disjunktion (`Or`) oder Implikation (`Implies`). Er kann aber auch unär (`Unary`) sein, also eine Negation (`Not`) oder der Existenzquantor (`Exists`). Außerdem kann ein Ausdruck ein Verweis auf ein Prädikat (`PredicateReference`) oder ein Objektauswähler (`ObjectSelector`) sein. Der Allquantor (`Forall`) ist ein spezieller Ausdruck, da er entsprechend der Definition der DRL mindestens zwei Objektauswähler enthalten muss.

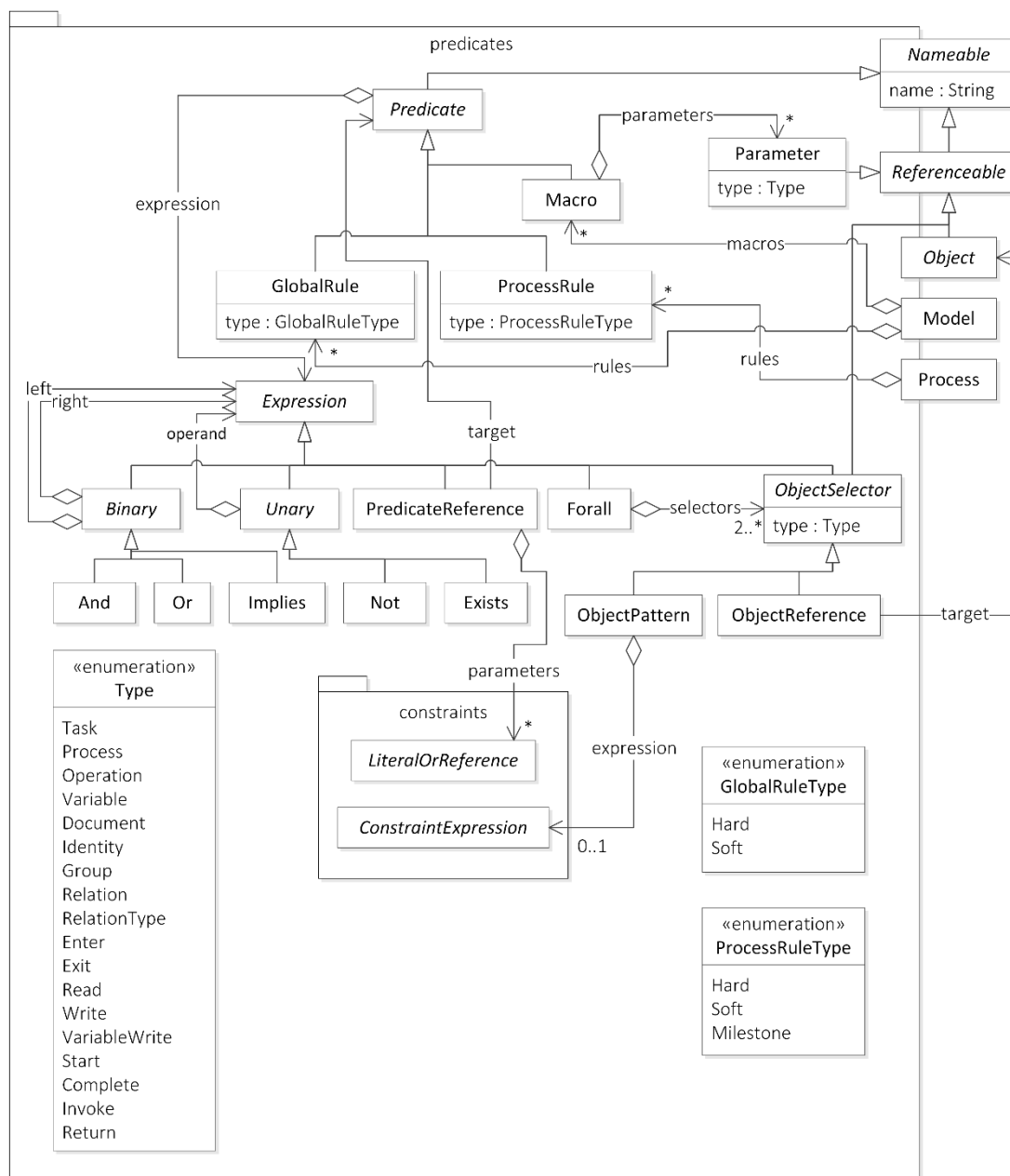


Abbildung 6.4 Abstrakte Syntax für Regeln in DPIL

Ein Objektauswähler bezieht sich immer auf einen bestimmten Element- oder Ereignistyp (**Type**). Er kann entweder ein Objekt direkt referenzieren (**ObjectReference**) oder mehrere anhand ihrer Eigenschaften eingrenzen (**ObjectPattern**).

Die in Abbildung 6.5 gezeigten Objektmuster (**ObjectPattern**) dienen dazu, Objekte anhand ihrer Eigenschaften auszuwählen. Dies ist notwendig, um zum Beispiel die Start-Ereignisse einer Aktivität zu selektieren, um sie anschließend einschränken oder von Bedingungen abhängig zu machen. Zur Auswahl von Objekten kann das Objektmuster mit einer Einschränkung (**ConstraintExpression**) versehen werden. Dabei handelt es sich um den Zugriff auf eine Objekteigenschaft (**PropertyAccess**) oder die Konjunktion (**ConstraintAnd**) oder Diskjunktion

(*ConstraintOr*) solcher Zugriffe. Der Zugriff auf eine Objekteigenschaft erfolgt über einen Eigenschaftsschlüssel (*PropertyKey*), der als Alias für die Attribute der Ereignisse dient. Der Schlüssel *Of* wird zum Beispiel auf das Attribut *element* der Ereignisbasisklasse *Event* abgebildet. Der Schlüssel *This* ist dabei ein spezieller Schlüssel, der sich auf das momentan ausgewählte Objekt selbst bezieht und nicht auf ein Attribut abgebildet wird. Ein Zugriff kann entweder den Wert einer Objekteigenschaft an eine Variable binden (*PropertyBinding*) oder ihn einschränken (*PropertyRestriction*). Durch eine Bindung kann der Wert einer Objekteigenschaft in anderen Teilausdrücken wiederum zur Einschränkung verwendet werden. Einschränkungen geschehen durch den Vergleich einer Objekteigenschaft mit einer Zeichenkette (*StringLiteral*), einer Zahl (*NumberLiteral*) oder einem Verweis (*Reference*). Als Ziel des Verweises kommen Variablen und Modellelemente in Frage.

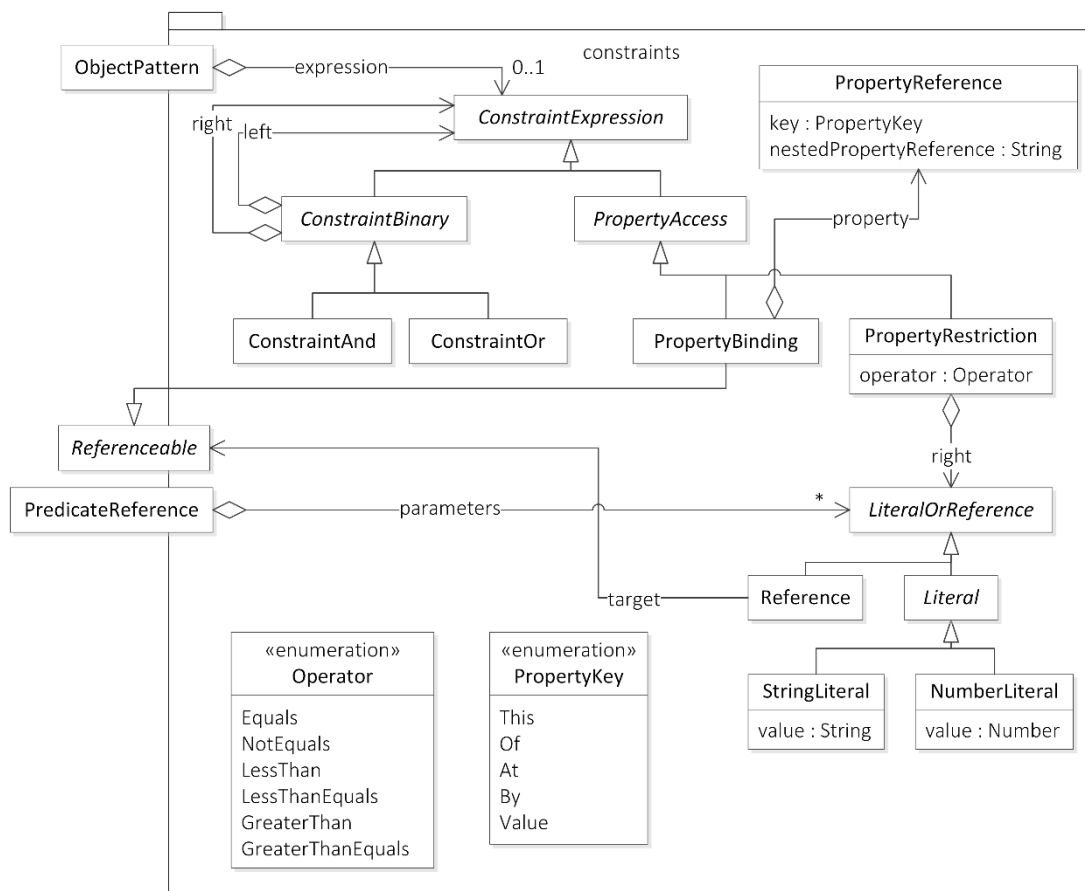


Abbildung 6.5 Abstrakte Syntax für Einschränkungen auf Objekten

6.4 Integrierte konkrete Syntax zur regelbasierten Prozessmodellierung

In den Abschnitten 6.1 bis 6.3 werden verschiedene Aspekte des Metamodells der Sprache DPIL beschrieben, also ihre abstrakte Syntax. Im Folgenden wird nun eine konkrete Syntax beschrieben, mit der DPIL-Modelle in textueller Form repräsentieren lassen. Mit ihr lassen sich die strukturellen Elemente wie Aktivitäten und Datenobjekte sowie die Prozessregeln integriert beschreiben. Zur Beschreibung der konkreten Syntax dient eine Grammatik in der Erweiterten Backus-Naur-Form

(EBNF) gemäß der Spezifikation der XML (W3C n.d.). Jede Regel der Grammatik definiert ein Symbol in der Form

Symbol := Ausdruck

Symbole werden mit einem Großbuchstaben begonnen, wenn sie das Startsymbol einer regulären Sprache darstellen, andernfalls werden sie mit einem Kleinbuchstaben begonnen. Feste Zeichenfolgen werden in Anführungszeichen geschrieben. Innerhalb des Ausdrucks auf der rechten Seite werden die folgenden Ausdrücke benutzt um Zeichenketten zu beschreiben:

[a-zA-Z] Jedes Zeichen zwischen dem angegebenen Bereich (inklusive)

[abc] Jedes Zeichen aus den aufgezählten

'**string**' Die feste Zeichenkette innerhalb der Anführungszeichen.

Diese Symbole können wie folgt zu komplexeren kombiniert werden wobei A und B einfache Ausdrücke sind:

(Ausdruck) Ausdruck wird als Einheit behandelt und kann wie unten kombiniert werden.

A? A oder nichts (A ist optional)

A B A gefolgt von B. Dieser Operator hat eine höhere Priorität als die Auswahl, so dass A B | C D identisch zu (A B) | (C D) ist.

A | B A oder B

A+ Ein oder mehrere Vorkommnisse von A. Die Verkettung hat eine höhere Priorität als die Auswahl, so dass A+ | B+ identisch zu (A+) | (B+) ist.

A* Kein oder mehrere Vorkommnisse von A. Die Verkettung hat eine höhere Priorität als die Auswahl, so dass A* | B* identisch zu (A*) | (B*) ist.

6.4.1 Modelle

Die folgenden Anteile der konkreten Syntax beziehen sich auf die abstrakte Syntax in Abschnitt 6.1. Im Kopf eines Modells werden die verwendeten Identitäten, Gruppen, Beziehungstypen aus dem Organisationsmodell aufgezählt, damit ihre Namen im Modell verfügbar und referenzierbar sind. Außerdem können die benötigten Verbindungen zu CMIS-kompatiblen ECM-Systemen angegeben werden, damit sie als Quelle oder Ziel von Datenobjekten genutzt werden können. Es können hier auch Makros und globale Regeln definiert werden. Ein Modell kann schließlich einen Wurzelprozess definieren. Auch ein Modell, das keinen Prozess definiert, ist sinnvoll. Es zählt dann nur Elemente des Organisationsmodells auf, definiert Makros oder Verbindungen zu ECM-Systemen und kann damit als Bibliothek für andere Modelle dienen.

```

Model:      Identity* Group* RelationType* Repository* Macro* GlobalRule*
           Process?
Identity:    'identity' ID
Group:       'group' ID
RelationType: 'relationtype' ID
Repository:  'repository' ID '{'
           'url' STRING
           'user' STRING
           'password' STRING
           'id' STRING '}'

```

6.4.2 Strukturelle Elemente

Ein Prozess (**process**) kann wiederum Aktivitäten, Datenobjekte und Prozessregeln enthalten. Aktivitäten und Datenobjekte müssen jeweils eine Kennung besitzen und können daneben einen Namen haben. Eine Aktivität ist ein wiederum Prozess, eine personelle Aufgabe (**task**) oder eine Operation (**operation**). Eine Operation hat keinen Namen sondern nur eine Kennung und einen Rumpf. Außerdem können den Formalparametern der Operation Datenobjekte als Aktualparameter zugeordnet werden und außerdem ein Datenobjekt für den Rückgabewert (**to**). Für Datenobjekte kann angegeben werden, ob es sich um eine Sammlung handelt (**collection**). Für ein Dokument kann die Anfrage für den Vorgabewert (**default**) und muss die Kennung der Verbindung (**at**) angegeben werden.

```

Process:     'process' ID STRING? '{'
           Activity*
           DataObject*
           ProcessRule* '}'
Activity:    Process | Task | Operation
Task:        'task' ID STRING?
Operation:    'operation' ID STRING
           ('{' Parameter (',' Parameter)* '}')?
           ('to' ID)
Parameter:    ID ID
DataObject:    Variable | Document
Variable:      'variable' 'collection'? ID STRING?
Document:      'document' 'collection'? ID STRING?
           ('default' STRING)?
           'at' ID

```

6.4.3 Regeln

Nach der konkreten Syntax der strukturellen Elemente folgen nun die Anteile der konkreten Syntax zu den Elementen in Abschnitt 6.3. Globale Regeln können harte Regeln (**ensure**) oder weiche Regeln (**advise**) sein. Eine Prozessregel kann außerdem ein Meilenstein (**milestone**). Regeln können eine Kennung und eine Beschreibung besitzen, die vom Rumpf durch einen Doppelpunkt abgetrennt werden müssen. Ein Makro hat eine Kennung und kann über Formalparameter verfügen. Der Rumpf des Makros wird durch **iff** abgetrennt.

GlobalRule: GlobalRuleType (ID? STRING? ':')? Expression
 GlobalRuleType: 'ensure' | 'advise'
 ProcessRule: ProcessRuleType (ID? STRING? ':')? Expression
 ProcessRuleType: 'ensure' | 'advise' | 'milestone'
 Macro: ID ('(' ID (',' ID) * ' '))? 'iff' Expression

Das Metamodell eines Regelausdrucks sieht einen Baum aus unären und binären Ausdrücken vor. Diese Schachtelung muss beim Entwurf der Grammatik berücksichtigt werden. Die Grammatik realisiert für alle Operatoren eine Infix-Notation, der Operator steht also zwischen den Operanden. Außerdem gibt die Grammatik die Operatorrangfolge vor, die in Tabelle 6.1 gezeigt ist. Demnach hat die Klammerung, Negation usw. den höchsten und die Implikation den niedrigsten Rang.

Tabelle 6.1 Operatorrangfolge

Operatoren	Rang
() not exists forall PredicateReference ObjectSelector	1
and	2
or	3
implies	4

Der Allquantor (**forall**) fordert für alle vom ersten Muster erfassten Objekte alle weiteren Muster. Eine Prädikatreferenz ist die Kennung des referenzierten Prädikats (Makro oder Regel) und die Liste der Aktualparameter. Objektauswähler wählen Objekte entweder anhand ihrer Kennung (ObjectReference) oder anhand anderer beliebiger Eigenschaften aus (ObjectConstraint). Die ausgewählten Objekte können durch einen Doppelpunkt an einer Variable gebunden werden.

Expression: Implies
 Implies: Or ('implies' Or)*
 Or: And ('or' And)*
 And: Unary ('and' Unary)*
 Unary: '(' Expression ')' | 'not' Unary | 'exists' Unary | Forall | PredicateReference | ObjectSelector
 Forall: 'forall' '(' ObjectSelector ObjectSelector+ ')'
 PredicateReference: ID '(' (LiteralOrReference (',' LiteralOrReference) * ' '))?
 ObjectSelector: ObjectConstraint | ObjectReference
 ObjectReference: Type ID (':' ID)?
 ObjectConstraint: Type '(' (ConstraintExpression ' '))? (':' ID)?
 Type: 'task' | 'operation' | 'process' | 'start' | ...

Sollen Objekte nicht anhand ihrer Kennung, sondern anhand von anderen Eigenschaften ausgewählt werden, so ist ein `ObjectConstraint` mit einer entsprechenden `ConstraintExpression` notwendig. Diese hat wieder die Struktur eines Ausdrucks, bildet also einen Baum aus unären und binären Teilausdrücken. Operatoren werden wieder in Infix-Notation realisiert und die Rangfolge entspricht der eines Regelausdrucks. Die Eigenschaften können entweder an Variablen gebunden werden (`PropertyBinding`), oder mit Variablen oder Konstanten verglichen werden (`PropertyRestriction`).

```

ConstraintExpression: ConstraintOr
ConstraintOr:      ConstraintAnd ('or' ConstraintAnd)*
ConstraintAnd:      ConstraintUnary ('and' ConstraintUnary)*
ConstraintUnary:     PropertyBinding
                   | PropertyRestriction
                   | '(' ConstraintExpression ')'
PropertyBinding:     PropertyReference ':' ID
PropertyReference:   PropertyKey RQID?
PropertyKey:         'this' | 'of' | 'by' | ...
PropertyRestriction: PropertyReference Operator? LiteralOrReference
Operator:            '=' | '!=' | ...
LiteralOrReference:  STRING | NUMBER | ID

```

Das Terminal `ID` wird für alle eindeutigen Kennungen verwendet, `RQID` für relative qualifizierte Bezeichner (zum Beispiel „`target.value`“), `STRING` für konstante Zeichenketten und `NUMBER` für konstante dezimale Zahlen mit einem Punkt als Dezimaltrennzeichen.

```

ID:      '^?('a'..'z'|'A'..'Z'|'_'|'_'|'0'..'9')*'
RQID:    '.' ID ('.' ID)*
STRING:  '"' ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') |
!('\\'|'"') )* '"'
NUMBER:  ('0'..'9')+ ('.' ('0'..'9')+)?

```

6.4.4 Beispiel

Das folgende Beispiel enthält viele der Schlüsselwörter und Konstruktionen der oben beschriebenen Grammatik:

```

use identity Alice
use group Secretariat
use relationtype memberOf

use repository btn4nv02 {
  url "http://btn4nv02.inf.uni-bayreuth.de:8080/chemistry/atom11"
  user "test"
  password "test"
  id "bpm"
}

```

```

consumes(consumer, incoming) iff
  start(of consumer) implies write(of incoming)

produces(producer, outgoing) iff
  complete(of producer) implies write(of outgoing)

process Trip "Business Trip" {
  task Apply "Apply for Business Trip"
  task Review "Review Business Trip Application"
  task Accept "Accept Business Trip Authorization"
  task Transfer "Book Transfer"
  task Accomodation "Book Accomodation"

  document Form "Application Form Template" default "/reiseant.doc" at
  btn4nv02
  document Application "Application" at btn4nv02
  variable Feedback "Review Feedback"

  ensure produces(Apply, Application)
  ensure consumes(Review, Application)
  ensure not write(of Form)

  advise "Application should be reviewed by secretariat":
    start(of Review by :i)
    implies relation(subject i predicate memberOf object Secretariat)

  advise "Transfer should be booked by applicant":
    start(of Transfer by :i) and start(of Apply)
    implies start(of Apply by i)

  advise "Accomodation should be booked by applicant":
    start(of Accomodation by :i) and start(of Apply)
    implies start(of Apply by i)

  milestone "Trip authorized": complete(of Accept)
  milestone "Transfer booked": complete(of Transfer)
  milestone "Accomodation booked": complete(of Accomodation)
}

```

Zunächst werden die Gruppe `Secretariat` und der Beziehungstyp `memberOf` aus der Organisationsverwaltung referenziert, um sie im Modell verwenden zu können. Anschließend wird die Verbindung zu einem CMIS-kompatiblen ECM-System definiert, so dass der Prozess auf Dokumente in diesem Repositorium zurückgreifen kann. Außerdem werden im Kopf des Modells die Makros `consumes` und `produces` definiert. Das Makro `consumes` fordert einen Schreibereignis (einen Wert) für ein Datenobjekt bevor eine Aktivität begonnen werden kann, es realisiert also eine eingehende Datenabhängigkeit. Das Makro `produces` fordert einen Wert für ein Datenobjekt bevor eine Aktivität abgeschlossen werden kann, es realisiert also eine

ausgehende Datenabhängigkeit. Das Modell beschreibt den Prozess einer Dienstreiseabwicklung und teilt ihn in die Aktivitäten **Apply** (Dienstreise beantragen), **Review** (Antrag prüfen), **Accept** (Genehmigung annehmen), **Transfer** (Transfer buchen) und **Accommodation** (Unterkunft buchen) ein. Außerdem sind am Prozess die Dokumente **Form** (Vorlage für den Antrag) und **Application** (Antrag) und die Variable **Feedback** (Rückmeldung zur Prüfung des Antrags). Es drei folgenden harten Regeln (**ensure**) fordern, dass aus der Beantragung der Antrag hervorgehen muss, dass zur Prüfung der Antrag vorliegen muss und dass die Vorlage für den Antrag nicht geändert werden darf. Die drei folgenden weichen Regeln (**advise**) empfehlen, dass der Antrag von einem Mitarbeiter des Sekretariats geprüft werden sollte und dass Transfer und Unterkunft vom Antragssteller gebucht werden sollten. Es ist jeweils ein Meilenstein erreicht, wenn der Antrag genehmigt wurde und der Transfer und die Unterkunft gebucht wurden.

6.5 Generierung von DRL aus dem DPIL-Modell

In einem DPIL-Modell lässt sich ein Baum aus zusammengesetzten Prozessen definieren. Für jedes Modell, also für jeden Baum wird eine Indexdatei im XML-Format generiert, die die strukturellen Elemente des Modells, also alle Elemente außer den Regeln, enthält. Für jeden zusammengesetzten Prozess wird außerdem eine Regeldatei in DRL-Syntax erzeugt, die die Regeln für die Elemente direkt unterhalb des Prozesses enthält. Alle Dateien eines Modells werden schließlich in einer komprimierten Archivdatei verpackt.

Tabelle 6.2 Wichtige Transformationen bei der Generierung von DRL aus DPIL

	DPIL	DRL
①	task : <i>t</i>	\$t : HumanTask()
②	task A : <i>a</i>	\$a : HumanTask(id == "A")
③	start (of A)	\$a : HumanTask(id == "A") and Start(element == \$a)
④	start (of : <i>x</i>)	Start(\$x : element)
⑤	advise "Message" Id: <i>expr</i>	rule Id @type("soft") @message("Message") when <i>expr</i> then listener.onRuleOccured(drools.getRule()); end
⑥	<i>x</i> implies <i>y</i>	not (<i>x</i> and not <i>y</i>)

Die wichtigsten Teilausdrücke bei der Transformation von DPIL zu DRL sind in Tabelle 6.2 aufgeführt. Der **ObjectConstraint** ohne Muster in ① erfasst alle Objekte vom Typ **HumanTask** und bindet sie an die Variable \$t. Die **ObjectReference** in ② referenziert Aufgabe A direkt und wird in das entsprechende Muster in DRL übersetzt, wobei die identifizierende Eigenschaft stets **id** heißt. Der Ausdruck in ③ bezieht sich ebenfalls auf die Aufgabe A ohne diese jedoch explizit durch ein Muster zu erfassen. Der Verweis auf den Schlüssel A wird in das entsprechende Muster

in DRL übersetzt. Der Vergleich der `element`-Eigenschaft bezieht sich dann auf die entsprechende Variable. Statt auf ein Element des Modells zu verweisen kann die Eigenschaft wie in ④ gezeigt auch an eine Variable gebunden werden. Die Bindung muss während der Transformation nach DRL syntaktisch entsprechend umgekehrt werden. DPIL-Regeln wie in ⑤ werden in DRL-Regeln übersetzt wobei die Kennung übernommen wird. Wurde im DPIL-Modell keine Kennung vergeben, so wird für die DRL-Regel ein Surrogat erzeugt. Der Typ der Regel und der Hinweistext werden in Attribute der Regel überführt. Die Bedingung der DRL-Regel ist der Ausdruck selbst, die Konsequenz ist die Meldung der erfüllten Regel an die RPVM. Da DRL keine Implikation unterstützt, muss eine Implikation in DPIL wie in ⑥ in den äquivalenten Ausdruck in DRL übersetzt werden. An diesen Beispielen wird auch deutlich, dass ein Prozessmodell in der DPIL-Notation deutlich kompakter formuliert werden kann, als in DRL.

6.6 Sprachinfrastruktur und integrierte Entwicklungsumgebung für DPIL-Prozesse

Die oberen Abschnitte beschreiben sowohl eine abstrakte, als auch eine konkrete Syntax für die Prozessmodellierungssprache DPIL. Auf dieser Grundlage muss nun ein Parser entwickelt werden, der einen Prozess in der DPIL-Notation entgegennimmt und in ein programmatisch interpretierbares Modell überführt. Zudem können abstrakte und konkrete Syntax genutzt werden, um den Entwurf von DPIL-Prozessen durch syntaktische, semantische und qualitative Prüfung zu unterstützen.

Das *Xtext*-Rahmenwerk kombiniert beide Aspekte. Es unterstützt den Bau der Sprachinfrastruktur, also Parser, Linker, Compiler und Interpreter und zusätzlich die Integration in die *Eclipse*-Plattform (Eclipse Foundation 2014). Auf der Basis von *Xtext* wird also eine integrierte Entwicklungsumgebung für DPIL-Prozesse (DPIL IDE) entwickelt.

6.6.1 Grundlegende Merkmale als Entwicklungsumgebung

Das menschliche visuelle System kann Farben sehr schnell wahrnehmen (Mackinlay 1986) und etwa dreimal schneller zwischen Farben unterscheiden als zwischen Formen (Lohse 1993). Die Syntaxhervorhebung (engl. *syntax highlighting*) nutzt genau diese Erkenntnisse und hebt bestimmte Schlüsselwörter und Zeichenkombinationen je nach ihrer Bedeutung hervor. *Xtext* erstellt die Syntaxhervorhebung auf der Basis der DPIL-Grammatik. Dieses typische Merkmal von Entwicklungsumgebungen kommt daher auch in der DPIL IDE zum Einsatz.

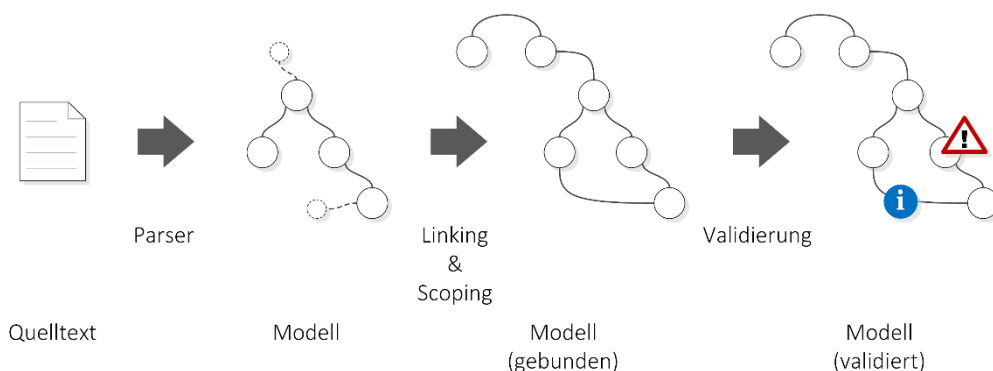


Abbildung 6.6 Aufgaben der Sprachinfrastruktur

Abbildung 6.6 zeigt die Aufgaben der Sprachinfrastruktur für DPIL. Die IDE hat zunächst die Aufgabe, aus dem Quelltext durch den Parser eine Instanz des in beschriebenen Modells zu erzeugen. Während dem Binden (engl. *linking*) werden Querverweise innerhalb des Modells aufgelöst. Es wird also zum Beispiel der Name einer verwendeten Variable durch einen Verweis auf diese Variable ersetzt. Dabei müssen Geltungsbereiche von Namen beachtet werden (engl. *scoping*). Auf der Basis der Geltungsbereiche kann auch eine Autovervollständigung (engl. *content assist*) angeboten werden. Außerdem werden diverse automatische Restrukturierungen (engl. *refactoring*) angeboten. Der Inhalt einer Quelltextdatei wird in der Übersichtsansicht (engl. *outline*) in einem Baum dargestellt.

6.6.2 Syntaktische, semantische und qualitative Überprüfung

Xtext generiert einen Parser auf der Grundlage der konkreten Syntax von DPIL. Syntaktische Fehler werden also zwangsläufig automatisch vom Parser erkannt und gemeldet. Die Suche nach semantischen Fehlern hingegen muss für DPIL spezifisch implementiert werden. Die folgenden Punkte bilden die wichtigsten semantischen Überprüfungen:

Unbekannte Verweisziele: Wird auf Modellelemente oder Variablen verwiesen, die nicht definiert bzw. nicht gebunden wurden, so wird der entsprechende Verweis als fehlerhaft markiert.

Geltungsbereich von Variablen: Gebundene Variablen gelten jeweils nur im entsprechenden Teilausdruck. Darüber hinaus können in manchen Ausdrücken keine Variablen gebunden werden. So ist die folgende Regel zum Beispiel nicht zulässig, weil innerhalb eines **exists**-Ausdrucks keine Variablen gebunden werden können:

```
advise exists start(of A by :i) and ...
```

Auch der folgende Ausdruck ist nicht zulässig, weil der Geltungsbereich einer Variable nicht über eine Disjunktion hinweg reicht:

```
advise start(of A by :i) or complete(of A by :i)
```

Geltungsbereich von Modellelementen: Modellelemente gelten nur in der Schachtelungshierarchie abwärts. Ein Datenobjekt kann zum Beispiel nicht in einem Prozess oberhalb des definierenden Prozesses referenziert werden. Im folgenden Modell ist der Verweis auf D in Q zum Beispiel nicht zulässig:

```
process P {
  process Q {
    document D
  }
  advise write(of D)
}
```

Darüber hinaus wird das Modell auf Konstruktionen untersucht, die zwar nicht zu Fehlern führen, aber die Qualität des Modells beeinträchtigen. Dazu zählen die folgenden Fälle:

Empfehlungen ohne einen Hinweistext: Der Hinweistext wird vom PAS verwendet um während der Ausführung von Aktionen abzurufen und dabei den Zusammenhang mit dem Prozessmodell herzustellen. Fehlt der Hinweistext, so kann hier dem Prozessteilnehmer nur die Kennung der Regel, die u.U. ein Surrogat ist, angezeigt werden.

Fehlende Meilensteine: Ein Modell, das keine Meilensteine definiert, gilt nie als beendet. Dies ist ein seltener Fall und wird daher zur Sicherheit markiert.

Redundante Ausdrücke: Wenn ein Teilausdruck als Makro definiert wurde, dann sollte ab diesem Punkt das Makro verwendet werden und nicht mehr der Ausdruck selbst. Dies garantiert, dass entsprechende Änderungen nur an einem Punkt durchgeführt werden müssen.

Ungenutzte Variablen: Variablen, die gebunden aber nicht genutzt werden, werden als ungenutzt und überflüssig markiert.

6.7 Zusammenfassung

Da die bestehenden Sprachen zur regelbasierten Prozessmodellierung die gestellten Anforderungen nicht erfüllen können, wurde im Rahmen dieser Arbeit eine entsprechende Sprache entworfen. Sie deckt die grundlegenden Perspektiven der Prozessmodellierung, also die Funktions-, Verhaltens-, Informations- und Organisationsperspektive ab und ermöglicht eine perspektivenübergreifende Modellierung. Außerdem unterstützt sie eine qualitative Unterscheidung von verpflichtenden und empfohlenen Handlungen. Für die strukturellen Anteile der Sprache wurde ein entsprechendes Metamodell entworfen. Dabei wurden vom Modellierungsprinzip unabhängige Anteile aus bestehenden Sprachen übernommen. Für die Formulierung und Auswertung der Prozessregeln wurde das Drools-Framework integriert. Da sich gezeigt hat, dass die Syntax des Frameworks für die Prozessmodellierung ungeeignet ist, wurde zudem eine spezifische Syntax entwickelt. Diese wird zur Ausführung in die von Drools verwendete Syntax übersetzt. In der entwickelten Syntax lassen sich sowohl die strukturellen Anteile (Aktivitäten, Datenobjekte usw.) eines Modells, als auch die Prozessregeln kombiniert beschreiben. Da es sich um eine textuelle Syntax ohne grafische Repräsentation handelt, ist die Sprache nicht für den Endanwender geeignet und wird daher als Zwischensprache (DPIL) bezeichnet. Zur Modellierung von DPIL-Prozessen wurde außerdem eine entsprechende Entwicklungsumgebung entwickelt, die die Modellierung durch syntaktische, semantische und qualitative und andere Merkmale unterstützt. Eine umfassende Evaluierung des Sprachumfangs von DPIL folgt in Abschnitt 7.

7 Realisierung von Prozessmustern in DPIL und Abbildung von anderen Sprachen

In Anlehnung an die Entwurfsmuster objektorientierter Systeme von Gamma et al. (Gamma et al. 1994) bilden die *Workflow Patterns* häufig wiederkehrende Muster in Prozessen, mit denen sich Anforderungen an ein PAS formulieren lassen. Mit Hilfe dieser Muster lassen sich PAS hinsichtlich ihrer Eignung und ihrer Ausdrucksstärke vergleichen. Die Muster sind unabhängig von einer Implementierung und von den spezifischen Anforderungen einer Domäne (van der Aalst et al. 2003).

Im Folgenden wird gezeigt, wie sich die Prozessmuster aus den Bereichen Kontrollfluss, Organisation und Daten in der Sprache DPIL in Verbindung mit der PN-Plattform umsetzen lassen. Das Ergebnis dient als Evaluation des regelbasierten Anteils der PN-Plattform. Außerdem wird die jeweilige Bewertung mit der von BPMN als Vertreter der prozeduralen Modellierung verglichen.

7.1 Realisierung kontrollflussbezogener Muster

Die Muster des Kontrollflusses (van der Aalst et al. 2003) sind durch den Vergleich klassischer prozeduraler WfMS motiviert, verweisen also meist ausdrücklich auf die klassischen Elemente prozeduraler Sprachen und sind daher nur bedingt auf eine regelbasierte Modellierung anwendbar.

Tabelle 7.1 Unterstützung von Mustern des Kontrollflusses durch DPIL und BPMN (vgl. (Workflow Patterns Initiative 2011))

Einfacher Kontrollfluss		D	B				
				34	Statische partielle Zusammenführung mehrerer Instanzen	-	+/-
1	Sequenz	+	+	35	Unterbrechende partielle Zusammenführung mehrerer Instanzen	-	+/-
2	Parallele Verzweigung (AND)	~	+	36	Dynamische partielle Zusammenführung mehrerer Instanzen	-	-
3	Synchronisation (AND)	+	+	Zustandsbasierte Muster			
4	Exklusive Verzweigung (XOR)	+	+	16	Aufgeschobene Auswahl	+	+
5	Einfache Zusammenführung (OR)	~	+	17	Verschachtelte parallele Führung	+	-
Komplexe Verzweigung und Synchronisation		D	B	18	Meilenstein	+	-
6	Mehrfachauswahl (OR)	+	+	39	Kritischer Abschnitt	-	-
7	Strukturierte synchronisierende Zusammenführung	?	+	40	Verschachtelte Führung	+	+/-

8	Mehrfache Zusammenführung	?	+	Abbruch			D	B
9	Strukturierter Diskriminator	?	+/-	19	Aufgabe abbrechen	-	+	
28	Blockierender Diskriminator	?	+/-	20	Prozess abbrechen	+	+	
29	Unterbrechender Diskriminator	?	+	25	Bereich abbrechen	+/-	+/-	
30	Strukturierte partielle Zusammenführung	?	+/-	26	Aktivität aus mehreren Instanzen abbrechen	-	+	
31	Blockierende partielle Zusammenführung	?	+/-	27	Aktivität aus mehreren Instanzen abschließen	-	-	
32	Unterbrechende partielle Zusammenführung	?	+/-	Wiederholung			D	B
33	Verallgemeinerte UND-Zusammenführung	?	+	10	Beliebige Zyklen	+	+	
37	Lokale synchronisierende Zusammenführung	?	-	21	Strukturierte Schleife	+	+	
38	Übergreifende synchronisierende Zusammenführung	?	-	22	Rekursion	-	-	
41	Zusammenführung von Strängen	?	+	Beendigung			D	B
42	Verzweigung von Strängen	?	+	11	Implizite Beendigung	-	+	
Mehrere Instanzen		D	B	43	Explizite Beendigung	+	+	
12	Ohne Synchronisierung	-	+	Auslöser			D	B
13	Mit a priori Wissen zur Entwurfszeit	-	+	23	Transienter Auslöser	-	-	
14	Mit a priori Wissen zur Laufzeit	-	+	24	Persistenter Auslöser	+	+	
15	Ohne a priori Wissen zur Laufzeit	-	-					

Tabelle 7.1 zeigt die Unterstützung der Muster des Kontrollflusses durch DPIL im Vergleich zu BPMN. Die Muster ohne Entsprechung in DPIL (~) werden als unterstützt (+) gewertet. Die Muster, für die keine Entsprechung in DPIL gefunden werden konnte (?) werden als nicht unterstützt (-) gewertet. Muster mit teilweiser Unterstützung (+/-) zählen zu 50%. In Summe werden durch DPIL also 15,5 (36%) und von BPMN 28,5 (66%) von 43 Mustern abgedeckt.

7.1.1 Einfacher Kontrollfluss

Das Sequenzmuster (WCP1) fordert, dass eine Aktivität aktiviert wird, sobald eine andere abgeschlossen ist. Das folgende Modell fordert diesen Zusammenhang zwischen den Aktivitäten A und B:

```
sequence(a, b) iff  
  start(of b at :t) implies complete(of a at < t)
```

```
task CaptureCardDetails
```

```
task VerifyAccount
```

```
ensure sequence(CaptureCardDetails, VerifyAccount)
```

Das Ergebnis ist eine zeitliche Halbordnung, nicht zu vergleichen mit dem Sequenzfluss in BPMN (Object Management Group Inc. 2011).

Eine parallele Verzweigung (WCP2) (auch „UND-Verzweigung“) ist ein Punkt, an dem sich die Kontrolle auf mehrere parallele Pfade aufteilt, so dass Aktivitäten gleichzeitig oder in beliebiger Reihenfolge durchgeführt werden können. Dieses Muster hat keine Entsprechung in einem regelbasierten Modell, da ohne einschränkende Regeln alle Aktivitäten parallel durchgeführt werden können.

Eine exklusive Verzweigung (WCP4) (auch „XOR-Verzweigung“ oder „Entscheidung“) ist ein Punkt, an dem basierend auf einer Entscheidung oder Prozesskontrolldaten nur einer von mehreren Pfaden gewählt wird. Dies entspricht sich gegenseitig ausschließenden Eingangsbedingungen für Aktivitäten:

```
lessThan(variable, number) iff  
  write(of variable value < number at :t)  
  and not write(of variable at > t)
```

```
greaterThan(variable, number) iff  
  write(of variable value > number at :t)  
  and not write(of variable at > t)
```

```
greaterThanEquals(variable, number) iff  
  write(of variable value >= number at :t)  
  and not write(of variable at > t)
```

```
variable Volume
```

```
task DispatchBobcat
```

```
task DispatchBackhoe
```

```
task DispatchD9Excavator
```

```
ensure start(of DispatchBackhoe) implies lessThan(Volume, 10)  
ensure start(of DispatchBobcat) implies greaterThanEquals(Volume, 10)  
  and lessThan(Volume, 100)  
ensure start(of DispatchD9Excavator) implies greaterThanEquals(Volume, 100)  
  and lessThan(Volume, 1000)
```


Die Synchronisation (WCP3) (auch „UND-Zusammenführung“) mehrerer paralleler Pfade entspricht einer zeitlichen Halbordnung ähnlich der Sequenz. Folgendes Modell bildet eine Synchronisation vor Aktivität C:

task CheckInvoice

task ProduceInvoice

task DispatchGoods

ensure sequence(CheckInvoice, DispatchGoods)

and sequence(ProduceInvoice, DispatchGoods)

Für eine einfache Zusammenführung (WCP5) (auch „XOR-Zusammenführung“) gibt es wiederum keine Entsprechung in regelbasierten Modellen.

7.1.2 Komplexe Verzweigung und Synchronisation

Eine Mehrfachauswahl (WCP6) (auch „OR-Verzweigung“) entspricht sich nicht gegenseitig ausschließenden Eingangsbedingungen für Aktivitäten (siehe WCP4).

Die verbleibenden Muster WCP7 bis 9, 28 bis 33, 37, 38, 41 und 42 beschreiben verschiedene Interpretationen einer inklusiven Zusammenführung (auch „OR-Zusammenführung“) und weitere komplexe prozedurale Muster.

7.1.3 Mehrere Instanzen

Der Einfachheit halber unterstützt DPIL keine mehrfach instanziierten Aktivitäten. Die entsprechenden Muster WCP12 bis 15 und 34 bis 36 werden daher nicht unterstützt.

7.1.4 Zustandsbasierte Muster

Die Entscheidung zwischen zwei Teilprozessen lässt sich bis zum letztmöglichen Zeitpunkt aufschieben (WCP16) indem entsprechende Eingangsbedingungen verwendet werden. Sobald eine Aktivität gestartet wurde, soll die Zuweisung einer anderen zurückgezogen werden:

task ContactCustomer

task Escalate

ensure (**start**(of ContactCustomer at :tC) **implies not** **start**(of Escalate at < tC))

and (**start**(of Escalate at :tE) **implies not** **start**(of ContactCustomer at < tE))

Die verschachtelte parallele Führung (WCP17) fordert, dass eine Gruppe von Aktivitäten jeweils genau einmal und entsprechend einer zeitlichen Halbordnung durchgeführt werden müssen, sich aber nicht überschneiden dürfen. Die zeitliche Halbordnung kann durch die Sequenz geregelt werden. Die Anforderung, dass sich die Aktivitäten nicht überschneiden dürfen, wird durch eine weitere Regel realisiert:

serializeAll iff

start(of :t at :s) and not complete(of t at > s) implies not start(at > s)

task PickGoods

task PackGoods

task PrepareInvoice

ensure serializeAll

ensure sequence(PickGoods, PackGoods)

Ein Meilenstein (WCP18) ist ein Zustand, von dem die Aktivierung von Aktivitäten abhängig gemacht werden kann:

task EnrolStudent

task OpenEnrolment

task CloseOffEnrolment

milestone EnrolmentsAccepted: **complete**(of OpenEnrolment)

and not start(of CloseOffEnrolment)

ensure start(of EnrolStudent) **implies** EnrolmentsAccepted

Kritische Abschnitte (WCP39) sind zwei oder mehr Teilprozesse, die nicht nebenläufig durchgeführt werden dürfen, weil sie zum Beispiel auf eine gemeinsame Ressource zugreifen.

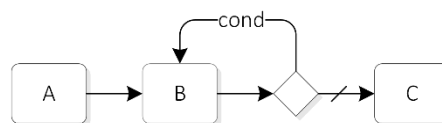
Das Interleaved Routing (WCP40) fordert, dass aus einer Menge von Aktivitäten jede genau einmal durchgeführt wird. Die Reihenfolge ist beliebig aber die Aktivitäten dürfen sich nie überlappen. Dies entspricht der WCP17 ohne zeitliche Ordnung.

7.1.5 Abbruch

Einzelne Aufgaben können nur entzogen werden (WCP19), solange sie noch nicht gestartet wurden. Eine Prozessinstanz wird abgebrochen (WCP20), wenn alle Meilensteine erfüllt sind. Hierbei werden auch gestartete Aufgaben entzogen. Mit diesem Mechanismus lassen sich auch Bereiche von Aktivitäten abbrechen (WCP25), wenn sie in einem Subprozess enthalten sind. Da keine mehrfach instanziierten Aktivitäten unterstützt werden, können auch die entsprechenden Abbruchmuster nicht unterstützt werden.

7.1.6 Wiederholung

Ohne Einschränkungen können alle Aktivitäten eines Prozess beliebig oft wiederholt werden (WCP10). Eine strukturierte Schleife (WCP21) ist ein klassisches prozedurales Muster. Sie lässt sich durch zeitlich ordnende Regeln und entsprechende Ein- und Austrittsbedingungen realisieren.



Dasselbe Verhalten lässt sich durch folgenden DPIL-Prozess erreichen:

```

once(t) iff
    start(of t at :p) implies not complete(of t at < p)

process WCP21 {
    task A
    task B
    task C

    ensure once(A)
    ensure sequence(A, B)
    ensure sequence(B, C)
    ensure start(of B) implies ...
    ensure start(of C) implies not ...

    milestone complete(of C)
}

```

7.1.7 Beendigung

Die Beendigung einer Prozessinstanz ist implizit (WCP11), wenn sie beendet wird, sobald keine Aktivitäten mehr durchgeführt werden können. Diese Form ist in DPIL nicht möglich, da der Beginn einer Aktivität von extern verwalteten Daten abhängen kann. DPIL unterstützt ausschließlich die explizite Form der Beendigung (WCP43) durch Meilensteine. Sobald alle definierten Meilensteine erfüllt sind, gilt die Prozessinstanz als beendet.

7.1.8 Auslöser

Ein Auslöser ist ein Signal aus einem anderen Teil des Prozesses oder aus der Umgebung, auf das mit der Aktivierung einer Aufgabe reagiert wird. Ein solcher Auslöser ist in jedem Fall persistent (WCP24), weil alle Mechanismen zur Umsetzung auf persistenten Ereignissen basieren. Im folgenden Beispiel wird die Aufgabe *CheckSensor* zugewiesen, sobald die Methode *MonitorAlarm* zurückkehrt:

```

operation MonitorAlarm "org.example.Alarm.monitor()"
task CheckSensor
ensure start(of CheckSensor) implies return(of MonitorAlarm)

```

7.2 Realisierung datenbezogener Muster

Neben dem Ablauf der Aktivitäten, also dem Kontrollfluss, gehört der Umgang mit Daten zu den klassischen Aspekten des Prozessmanagements. Die Art und Weise in der die an einem Prozess beteiligten Daten strukturiert und in den Prozess eingebunden werden können, wird in den datenbezogenen Prozessmustern sprachunabhängig charakterisiert. Die Muster gliedern sich in die Bereiche Sichtbarkeit von Daten, Interaktion mit Daten, Übergabe innerhalb des Prozesses und über seine Grenzen hinweg und der Einfluss von Daten auf den Ablauf (datenbasierte Führung) (Russell, Hofstede, et al. 2004).

Tabelle 7.2 Unterstützung von datenbezogenen Mustern durch DPIL und BPMN

Sichtbarkeit		D	B	21	Umgebung zu Instanz	+	-
1	Aufgabenbezogene Daten	+/-	+	22	Prozessinstanz zu Umgebung (Pull)	+	-
2	Blockbezogene Daten	+	+	23	Prozess zu Umgebung (Push)	+	-
3	Bereichsbezogene Daten	-	-	24	Umgebung zu Prozess (Pull)	-	-
4	Aufgabeninstanzbezogene Daten	-	+/-	25	Umgebung zu Prozess (Push)	-	-
5	Prozessinstanzbezogene Daten	+	+	26	Prozess zu Umgebung (Pull)	+	-
6	Ordnerbezogene Daten	+	-	Übergabe			D B
7	Prozessbezogene Daten	+	-	27	by value (eingehend)	-	+
8	Umgebungsdaten	+/-	-	28	by value (ausgehend)	-	+
Interner Austausch		D	B	29	copy in/copy out	-	+/-
9	Aufgabe zu Aufgabe	+	+	30	by reference (ungesperrt)	+	-
10	Blockaufgabe zu Subprozess	+	+	31	by reference (gesperrt)	+	+
11	Subprozess zu Blockaufgabe	+	+	32	Transformation (eingehend)	+	+/-
12	Zu mehrfach instanzzierbarer Aufgabe	-	-	33	Transformation (ausgehend)	+	+/-
13	Von mehrfach instanzzierbarer Aufgabe	-	-	Datenbasierte Führung			D B
14	Prozessinstanz zu Prozessinstanz	+	-	34	Vorbedingung für Aufgaben: Existenz von Daten	+	+
Externer Austausch		D	B	35	Vorbedingung für Aufgaben: Datenwert	+	-
15	Aufgabe zu Umgebung (Push)	+	+	36	Nachbedingung für Aufgaben: Existenz von Daten	+	+
16	Umgebung zu Aufgabe (Pull)	+	+	37	Nachbedingung für Aufgaben: Datenwert	+	-
17	Umgebung zu Aufgabe (Push)	+	+	38	Ereignisbasierter Auslöser für Aufgaben	+	+
18	Aufgabe zu Umgebung (Pull)	+	+	39	Datenbasierter Auslöser für Aufgaben	+	+
19	Prozessinstanz zu Umgebung (Push)	+	-	40	Datenbasierte Führung	+	+
20	Umgebung zu Prozessinstanz (Pull)	+	-				

Tabelle 7.2 zeigt die Unterstützung der datenbezogenen Muster durch DPIL und BPMN. DPIL unterstützt 30 (75%) und BPMN 20 (50%) von 40 Mustern.

7.2.1 Sichtbarkeit

Eine Variable ist innerhalb der Instanzen des umgebenden Prozesses (WDP5) und denen seiner Subprozesse (WDP2) sichtbar. Um zumindest ihre Sichtbarkeit auf eine einzelne Aufgabe zu begrenzen (WDP1), können die Lese- und Schreibereignisse entsprechend eingeschränkt werden:

```
local(task, object) iff
  read(of object at :tr) implies start(of task at < tr at :ts)
  and not complete(of task at > ts at < tr)
```

```
task CalculateFlightPath
variable WorkingTrajectory
```

```
advise local(CalculateFlightPath, WorkingTrajectory)
```

Die Lebensdauer der Variable bzw. ihres Werts umspannt aber die der Prozessinstanz und lässt sich auch nicht verkürzen.

Der Zugriff auf Datenobjekte lässt sich auch von der jeweiligen Prozessinstanz abhängig machen (WDP6). Im folgenden Beispiel kann abhängig vom Ordner, der in *SelectFolder* gewählt wird in *AccessFolder* auf eine andere Sammlung von Dokumenten zugegriffen werden:

```
task SelectFolder
task AccessFolder
```

```
document collection FolderA default "/A" at Local
document collection FolderB default "/B" at Local
```

```
variable FolderName
```

```
ensure produces(SelectFolder, FolderName)
ensure sequence(SelectFolder, AccessFolder)
```

```
ensure read(of FolderA) implies write(of FolderName value "A")
ensure read(of FolderB) implies write(of FolderName value "B")
```

Umgebungsdaten müssen zunächst durch den Aufruf einer Dienstmethode gelesen und in eine Variable geschrieben werden, damit sie im Prozess verfügbar werden:

```
operation GetTemperature "http://service.org/temperature" to Temperature
task CheckTemperature
```

```
variable Temperature
ensure start(of CheckTemperature) implies return(of GetTemperature)
```

7.2.2 Interaktion

Datenobjekte sind für alle Elemente des umgebenden Prozesses sichtbar. Ihre Werte können also zwischen den Aufgabeninstanzen eines Prozesses kommuniziert werden (WDP9). Da ein Datenobjekt auch für alle Subprozesse des umgebenden Prozesses sichtbar sind, können dessen Werte implizit an die Subprozesse übergeben (WDP10) und wieder genutzt werden (WDP11). Da es zu einer Aufgabe zur selben Zeit immer nur eine Instanz geben kann, wird eine Datenübergabe an mehrfach instanziierbare Aufgaben (WDP12, 13) nicht betrachtet. Die Datenübergabe zwischen mehreren parallelen Prozessinstanzen (WDP14) kann über extern verwaltete Dokumente realisiert werden.

Wie oben gezeigt, lassen sich Umgebungsdaten mittels Skripte übergeben (WDP15) und anfragen (WDP16). Daten können ungeplant während der Durchführung einer Aufgabe implizit empfangen werden (WDP17), indem extern verwaltete Dokumente verwendet und diese während der Durchführung verändert werden. Auf diese Weise können Daten auch wieder zurückgeliefert werden (WDP18). Da die Datenobjekte für alle Elemente des umgebenden Prozesses sichtbar sind gilt dies für die gesamte Prozessinstanz (WDP21, 22).

Werden ausschließlich extern verwaltete Dokumente in einem Prozess verwendet, dann werden die Datenwerte der entsprechenden Prozessinstanzen mit dem Dokumentenmanagementsystem synchronisiert und der Zustand der Prozessinstanz von außen sichtbar. Außerdem lässt sich ein entsprechender Skriptaufruf an das Ende einer Prozessinstanz stellen, um deren Daten an die Umgebung zu übergeben (WDP19). Umgekehrt lassen jederzeit Daten über extern verwaltete Dokumente von der Umgebung an die Prozessinstanz übergeben (WDP20).

Die PN-Plattform unterstützt die Überwachung von prozessbezogenen Kennzahlen, wie zum Beispiel die Anzahl der abgeschlossenen Instanzen eines Prozesses und kann diese Daten in regelmäßigen Abständen an ein angebundenes System übergeben (WDP23) oder sie zur Anfrage bereitstellen (WDP26).

7.2.3 Übergabe

Der Zugriff auf Datenobjekte geschieht grundsätzlich *by reference* (WDP30). Sperren lassen sich durch Einschränkungen auf Schreib- und Leseereignissen realisieren (WDP31). Die Transformation von Datenwerten (WDP32, 33) lässt sich in Dienstoperationen auslagern:

```
operation FromKmhToMph "http://service.org/kmhToMph" with SpeedKmh to
SpeedMph
task ReviewSpeed
variable SpeedKmh
variable SpeedMph
ensure start(of ReviewSpeed) implies return(of FromKmhToMph)
ensure invoke(of FromKmhToMph) implies write(of SpeedKmh)
```

7.2.4 Datenbasierte Führung

Vor- und Nachbedingungen für Aufgaben lassen sich durch die Einschränkung ihrer *Start*- und *Complete*-Ereignisse realisieren. Diese können entweder von der Existenz eines Datenwerts oder von einem bestimmten Datenwert abhängig gemacht werden (WDP34-37):

```
consumes(consumer, incoming) iff
    start(of consumer) implies write(of incoming)
```

```
produces(producer, outgoing) iff
    complete(of producer) implies write(of outgoing)
```

```
task RocketInitiation
```

```
variable Countdown
```

```
variable IgnitionData
```

```
ensure consumes(RocketInitiation, Countdown)
```

```
ensure start(of RocketInitiation) implies write(of Countdown value 2)
```

```
ensure produces(RocketInitiation, IgnitionData)
```

Aufgaben können durch die Umgebung angestoßen und Daten übergeben werden (WDP38):

```
task EmergencyShutdown
```

```
operation MonitorPowerAlarm "org.example.Alarm.monitor()" to AlarmCode
```

```
variable AlarmCode
```

```
ensure start(of EmergencyShutdown) implies return(of MonitorPowerAlarm)
```

Aufgaben lassen sich ebenso auf der Basis von bestimmten Datenwerten auslösen (WDP39):

```
task RebalancePortfolio
```

```
variable LoanMargin
```

```
ensure start(of RebalancePortfolio) implies write(of LoanMargin value >
0.85)
```

Eine datenbasierte Führung (WDP40) entspricht im regelbasierten Fall einer exklusiven Verzweigung (WCP4) bzw. einer Mehrfachauswahl (WCP6).

7.3 Realisierung organisatorischer Muster

Ursprünglich umfassten die Workflow Patterns ausschließlich die klassischen Perspektiven der Automatisierung, also Muster des Kontroll- und des Datenflusses. Die Einbindung menschlicher Teilnehmer rückte jedoch mehr und mehr in den Vordergrund. Die entsprechende Unterstützung durch Standards der Prozessmodellierung war unzureichend und so wurde das Rahmenwerk der Workflow Patterns um Muster der organisatorischen Perspektive erweitert. Dadurch ließen sich nun Systeme auch hinsichtlich ihrer Unterstützung für menschliche Prozessteilnehmer sprachunabhängig vergleichen (Russell, ter Hofstede, et al. 2004).

Tabelle 7.3 Unterstützung von organisatorischen Mustern durch DPIL und BPMN

Erzeugung		D	B	23	Teilnehmerinitiierte Ausführung (angebotene Aufgabe)	+	-
1	Direkte Verteilung	+	+	24	Systembestimmte Arbeitsliste	-	-
2	Rollenbasierte Verteilung	+	+	25	Teilnehmerbestimmte Arbeitsliste	+	-
3	Verzögerte Verteilung	+	-	26	Freie Wahl	+	-
4	Berechtigung	+	-	Abweichung		D	B
5	Trennung von Zuständigkeiten	+	-	27	Bevollmächtigung	-	-
6	Fallbehandlung	-	-	28	Eskalation	-	-
7	Bindung von Zuständigkeiten	+	-	29	Freigabe	+	-
8	Fähigkeitsbasierte Verteilung	+	-	30	Zustandsbehaftete Neuzuweisung	-	-
9	Verlaufsorientierte Verteilung	+/-	-	31	Zustandslose Neuzuweisung	-	-
10	Organisationsbasierte Verteilung	+	-	32	Unterbrechung / Fortsetzung	+	-
11	Automatische Durchführung	+	+	33	Überspringen	+/-	-
Push		D	B	34	Wiederholen	+	-
12	Verteilung durch Angebot (einzelne Ressource)	+	-	35	Vorarbeiten	+/-	-
13	Verteilung durch Angebot (mehrere Ressourcen)	+	-	Automatischer Beginn		D	B
14	Verteilung durch Zuweisung	+	+	36	Arbeitsbeginn bei Erzeugung	+	+
15	Zufällige Zuweisung	-	-	37	Arbeitsbeginn bei Zuweisung	-	-
16	Zuweisung nach Round-Robin-Verfahren	-	-	38	Gestapelte Ausführung	-	-
17	Zuweisung nach Shortest Queue-Verfahren	-	-	39	Verkettete Ausführung	+	+
18	Frühe Verteilung	-	-	Sichtbarkeit		D	B
19	Verteilung bei Aktivierung	+	+	40	Konfigurierbare Sichtbarkeit nicht zugewiesener Aufgaben	-	-

20	Späte Verteilung	-	-	41	Konfigurierbare Sichtbarkeit zugewiesener Aufgaben	+	-
Pull		D	B	Mehrere Teilnehmer		D	B
21	Teilnehmerinitiierte Zuweisung	+	-	42	Gleichzeitige Ausführung	+	+
22	Teilnehmerinitiierte Ausführung (zugewiesene Aufgabe)	+	-	43	Zusätzliche Teilnehmer	-	-

Tabelle 7.3 zeigt die Unterstützung der organisatorischen Muster durch DPIL und BPMN. DPIL unterstützt 26,5 (62%) und BPMN 8 (19%) von 43 Mustern.

Die einfachste Form der Verteilung ist die direkte Zuweisung einer Identität zur einer Aufgabe zur Entwurfszeit (WRP1). Aufgaben können aber auch an Rollen verteilt werden (WRP2). Rollen dienen dabei als Mittel zur Gruppierung von Identitäten mit ähnlichen Eigenschaften. Die entsprechende Aufgabe wird an alle Mitglieder der angegebenen Rolle(n) verteilt. Eine Zuweisung kann auch auf der Basis von bestimmten Fähigkeiten (WRP8) oder organisatorischen Beziehungen (WRP10) verteilt werden.

Das folgende Modell enthält alle drei Muster. Aufgabe *FixBentley* soll ausschließlich von *Fred* durchgeführt werden. Aufgabe *ApproveTravelRequisition* soll von einem *Manager* durchgeführt werden. *AirframeExamination* soll von einem *Engineer* durchgeführt, der mindestens 10 Jahre Erfahrung in der Wartung hat. *AuthoriseExpenditure* soll dem Manager desjenigen zugewiesen werden, der *ClaimExpenditure* durchgeführt hat.

```

use identity Fred
use group Manager
use group Engineer
use relationtype hasRole
use relationtype hasJob
use relationtype isManagerOf

```

```

direct(t, i) iff
  start(of t) implies start(of t by i)

```

```

role(t, r) iff
  start(of t by i)
  implies relation(subject i predicate hasRole object r)

```

```

process Organisation {
  task FixBentley
  task ApproveTravelRequisition
  task AirframeExamination
  task ClaimExpenditure

```

```

task AuthoriseExpenditure

advise direct(FixBentley, Fred)
advise role(ApproveTravelRequisition, Manager)

advise start(of AirframeExamination)
  implies
    start(of AirframeExamination
      by :i
      by.servicingExperienceInYears >= 10)
    and relation(subject i predicate hasJob object Engineer)

advise start(of ClaimExpenditure by :claimer)
  and start(of AuthoriseExpenditure by :authoriser)
  implies relation(subject authoriser
    predicate isManagerOf
    object claimer)
}

```

Die Zuständigkeiten für Aufgaben lassen sich trennen (WRP5) und verknüpfen (WRP7). Im folgenden Modell sollen *UmpireMatch* und *PrepareMatchReport* von denselben Identitäten durchgeführt werden, *PrepareCheque* und *CountersignCheque* aber von unterschiedlichen.

```

separate(a, b) iff
  start(of b by :ib) implies start(of a by != ib)

retain(a, b) iff
  start(of b by :ib) implies start(of a by ib)

process Organisation {
  task UmpireMatch
  task PrepareMatchReport
  task PrepareCheque
  task CountersignCheque

  advise retain(UmpireMatch, PrepareMatchReport)
  advise separate(PrepareCheque, CountersignCheque)
}

```

Eine Zuweisung lässt sich verzögern (WRP3), indem die Identität, der eine Aufgabe zugewiesen werden soll, erst zur Laufzeit des Prozesses durch den Wert eines Datenobjekts angegeben wird:

```

task AssessDamage
variable Performer

advise start(of AssessDamage) implies variablewrite(of Performer value :p)
  and start(of AssessDamage by.id p)

```

Eine verlaufsorientierte Zuweisung (WRP9) bezieht sich auf den bisherigen Verlauf der Prozessinstanz oder früherer Prozessinstanzen. Ein Bezug auf frühere Instanzen wird von DPIL der Einfachheit halber nicht unterstützt.

Aufgaben können ohne Verpflichtung einer einzelnen (WRP12) oder mehreren Identitäten (WRP13) angeboten werden indem die entsprechende Zuweisung als weiche Regel (*advise*) modelliert wird. Eine verbindliche Zuweisung an eine Identität (WRP14) kann dementsprechend mit einer harten Regel (*ensure*) realisiert werden.

Die PN-Plattform unterstützt keine Ablaufplanung (engl. *scheduling*) und daher keine der entsprechenden Muster (WRP15, 16 & 17).

Teilnehmer können Aufgaben reservieren, sie damit für andere Teilnehmer entsprechend markieren (WRP21) und die Arbeit im Anschluss daran beginnen (WRP22). Eine Aufgabe kann aber auch direkt gestartet werden, sobald sie angeboten wurde (WRP23).

Die Sortierung der Arbeitsliste lässt sich nicht durch das Prozessmodell (WRP24) aber durch die Prozess Teilnehmer (WRP25) steuern. Ein Prozess Teilnehmer kann mehrere Aufgaben gleichzeitig bearbeiten und selbst wählen, welche er als nächstes bearbeiten will (WRP26).

Aufgaben können nach einer Reservierung auch wieder freigegeben werden (WRP29) und stehen dann wieder allen potentiellen Bearbeitern zur Verfügung. Aufgaben können als unterbrochen markiert werden (WRP32). Aufgaben können nach dem Beginn umgehend abgeschlossen werden, sofern dies keiner Regel widerspricht und damit übersprungen werden (WRP33). Ohne einschränkende Regeln kann eine Aufgabe beliebig oft wiederholt werden (WRP34). Die Wiederholung kann dann eingeschränkt werden, wenn ein bestimmtes Ergebnis erreicht wurde.

Teilnehmer können in gewisser Weise vorarbeiten (WRP35), indem zum Beispiel eine zeitliche Ordnung zweier Aufgaben nur empfohlen aber nicht gefordert wird. Dem Teilnehmer wird dann die spätere Aufgabe unter Vorbehalt nur unter Vorbehalt zugewiesen bzw. angeboten.

Aufgaben werden gleichzeitig erzeugt und zugewiesen (WRP36), können aber nicht automatisch nach einer Zuweisung gestartet werden (WRP37). Nach Abschluss einer Aufgabe können unmittelbar weitere Aktivitäten angestoßen werden (WRP39).

Zugewiesene Aufgaben können allen Prozess Teilnehmern zusammen mit ihren aktuellen Bearbeitern angezeigt werden (WRP41). Ein Teilnehmer kann gleichzeitig mehrere Aufgaben bearbeiten (WRP42). Eine Aufgabe wird zur selben Zeit aber immer nur von einem Teilnehmer bearbeitet (WRP43).

7.4 Gegenseitige Ergänzung prozedurale und regelbasierter Modellierung

Die Evaluation zeigt zunächst, dass DPIL in Verbindung mit der PN-Plattform in Summe sogar rund 30% mehr Anforderungen an ein PAS erfüllt als BPMN. Sie zeigt aber auch, dass BPMN im Bereich der klassisch prozeduralen Muster DPIL mit 66% zu 36% Abdeckung überlegen ist. Stehen also genau spezifizierte komplexe Ablaufmuster im Vordergrund, ist BPMN zur Modellierung besser geeignet als DPIL. Datenbezogene und organisatorische Muster hingegen bilden oft perspektivenübergreifende Zusammenhänge, die sich DPIL besser modellieren lassen. Das

Ergebnis zeigt, dass sich die beiden Ansätze ergänzen und je nach Natur des Prozesses entschieden werden muss, nach welchem Prinzip er modelliert werden soll.

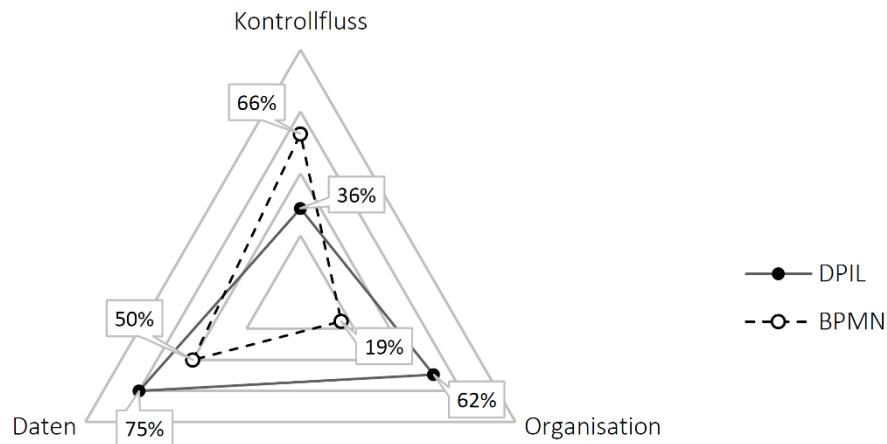


Abbildung 7.1 Zusammenfassung der Abdeckung von Prozessmustern durch DPIL und BPMN

7.5 Funktionale und operationale Muster

Neben der Verhaltens-, Informations- und Organisationsperspektive zählen auch die Funktions- und die Operationsperspektive zu den grundlegenden Perspektiven bei der Beschreibung von Prozessmodellen (Curtis 1992; Jablonski & Bußler 1996). Die Initiative um die *Workflow Patterns* sieht in diesen Bereichen jedoch keine gesonderten Muster vor.

Die Funktionsperspektive beschreibt den funktionalen Aufbau von Prozessmodellen. BPMN und DPIL unterscheiden gleichermaßen in zusammengesetzte und elementare Prozessschritte. Beide erlauben die Wiederverwendung von Teilprozessen durch Verweise auf untergeordnete Prozessmodelle. BPMN und DPIL werden daher bezüglich der Funktionsperspektive als gleichwertig betrachtet.

Die Operationsperspektive beschreibt die konkrete Realisierung eines elementaren Prozessschritts. Sowohl BPMN, als auch DPIL unterscheiden bei elementaren Prozessschritten grundsätzlich zwischen personellen Aufgaben und automatisierten Operationen. Auch bezüglich der Operationsperspektive können BPMN und DPIL daher als gleichwertig betrachtet werden.

7.6 Bibliothek von Modellierungsmustern

Einige der modellierungsbezogenen Prozessmuster aus den Abschnitten 7.1 bis 7.3 lassen sich in DPIL in Form von Makros kapseln. Da es sich bei den Mustern um häufige Muster der Prozessmodellierung handelt (van der Aalst et al. 2003), bilden die Makros in Tabelle 7.4 letztlich Vorschläge für die Modellierungskonstrukte einer höheren, auf DPIL basierenden Sprache. Die Makros können in einer grafischen Prozessmodellierungssprache zum Beispiel mit grafischen Elementen verknüpft werden.

Tabelle 7.4 Kandidaten für Konstrukte einer Modellierungssprache für Fachabteilungen

Kontrollflussbezogene Konstrukte	
<u>sequence</u> (<i>before</i> , <i>after</i>)	Die Aufgabe <i>after</i> kann erst begonnen werden, nachdem <i>before</i> abgeschlossen wurde.
<u>once</u> (<i>task</i>)	Die Aufgabe <i>task</i> kann nur gestartet werden, wenn sie nicht zuvor bereits einmal abgeschlossen wurde.
<u>serializeAll</u>	Die Aufgaben des Prozesses dürfen sich nicht zeitlich überlappen
Datenbezogene Konstrukte	
<u>local</u> (<i>task</i> , <i>object</i>)	Auf das Datenobjekt <i>object</i> kann nur zugegriffen werden, während die Aufgabe <i>task</i> bearbeitet wird.
<u>consumes</u> (<i>consumer</i> , <i>incoming</i>)	Die Aufgabe <i>consumer</i> kann erst gestartet werden, wenn ein Wert für das Datenobjekt <i>incoming</i> vorliegt.
<u>produces</u> (<i>producer</i> , <i>outgoing</i>)	Die Aufgabe <i>producer</i> kann erst abgeschlossen werden, wenn ein Wert für das Datenobjekt <i>outgoing</i> vorliegt.
Organisationsbezogene Konstrukte	
<u>direct</u> (<i>task</i> , <i>identity</i>)	Die Aufgabe <i>task</i> soll ausschließlich durch die Identität <i>identity</i> bearbeitet werden.
<u>role</u> (<i>task</i> , <i>role</i>)	Die Aufgabe <i>task</i> soll von einem Teilnehmer in der Rolle <i>role</i> bearbeitet werden.
<u>separate</u> (<i>task1</i> , <i>task2</i>)	Die Aufgaben <i>task1</i> und <i>task2</i> sollen von unterschiedlichen Identitäten bearbeitet werden.
<u>retain</u> (<i>task1</i> , <i>task2</i>)	Die Aufgaben <i>task1</i> und <i>task2</i> sollen von derselben Identität bearbeitet werden.

7.7 DPIL und DECLARE

Das Rahmenwerk DECLARE erlaubt eine regelbasierte Modellierung und Ausführung von Prozessen unter anderem auf der Basis linearer temporaler Logik (LTL). Da DECLARE derzeit eine wichtige Rolle in der Forschungsgemeinschaft des regelbasierten Prozessmanagements spielt, soll im Folgenden der Zusammenhang zwischen DPIL auf der einen und DECLARE auf der anderen Seite genauer beleuchtet werden.

Tabelle 7.5 Zentrale DECLARE-Regelvorlagen, ihre entsprechenden LTL-Formeln und ihre Bedeutung

Vorlage	LTL-Formel	Beschreibung
existence(A)	$\Diamond A$	A wird mindestens einmal ausgeführt.
init(A)	A	A ist die erste Aufgabe.
response(A, B)	$\Box(A \rightarrow \Diamond B)$	Immer wenn A ausgeführt wird, muss danach B ausgeführt werden. B muss nicht direkt nach A ausgeführt werden und A kann ein weiteres Mal zwischen dem ersten Mal und B ausgeführt werden.

precedence(A, B)	$(\neg B)WA$	B kann erst nach A ausgeführt werden. Andere Aufgaben können zwischen A und B ausgeführt werden.
succession(A, B)	$\text{response}(A, B) \wedge \text{precedence}(A, B)$	Konjunktion aus response und precedence
not co-existence(A, B)	$\neg(\Diamond A \wedge \Diamond B)$	A und B können nicht beide in derselben Prozessinstanz ausgeführt werden.
1 of 4(A, B, C, D)	$\Diamond A \vee \Diamond B \vee \Diamond C \vee \Diamond D$	Mindestens eine der vier Aufgaben muss ausgeführt werden.

Zur Ausführung werden die LTL-Formeln in Automaten übersetzt. Mit Hilfe dieser Automaten kann dann der Zustand einer Regel überwacht werden. DECLARE betrachtet dabei die Ausführung einer Aufgabe als unteilbares Ereignis, unterscheidet also nicht zwischen dem Start und dem Ende einer Aktivität. Jede Ausführung einer Aufgabe durch den Benutzer löst bei den Automaten einen Übergang aus. Befindet sich der Automat in einem akzeptierenden Zustand, gilt die Regel als *erfüllt*. Ist der Automat hingegen in einem nicht akzeptierenden Zustand, von dem aus jedoch ein akzeptierender Zustand erreichbar ist, dann gilt die entsprechende Regel als *vorübergehend verletzt*. Andernfalls ist und bleibt die Regel *verletzt*. Zudem unterscheidet DECLARE zwischen verbindlichen und optionalen Regeln. Für den Automaten einer verbindlichen Regel muss im Gegensatz zu dem einer optionalen Regel stets ein akzeptierender Zustand erreichbar sein; verbindliche Regeln müssen also stets erfüllbar sein.

Demnach können verbindliche DECLARE-Regeln auf harte DPIL-Regeln und optionale DECLARE-Regeln auf weiche DPIL-Regeln abgebildet werden. Die Übersetzung von DECLARE- auf DPIL-Regeln ist in Tabelle 7.6 gezeigt. Die unterschiedliche Formulierung liegt in der unterschiedlichen Semantik der beiden Ansätze begründet.

Tabelle 7.6 Übersetzung der zentralen DECLARE-Regelvorlagen in DPIL-Makros

DECLARE	DPIL
existence(A)	<u>existence</u> (A) iff complete(of A)
init(A)	<u>init</u> (A) iff start implies start(of A at :t) and not start(of != A at < t)
response(A, B)	<u>response</u> (A, B) iff complete(of A at :t) implies complete(of B at > t)
precedence(A, B)	<u>precedence</u> (A, B) iff start(of B at :t) implies complete(of A at < t)
succession(A, B)	<u>succession</u> (A, B) iff <u>response</u> (A, B) and <u>precedence</u> (A, B)
not co-existence(A, B)	<u>notCoExistence</u> (A, B) iff not(complete(of A at :t1) and start(of B at > t1)) and not(complete(of B at :t2) and start(of A at > t2))

1 of 4(A, B, C, D)	$_1of4(A, B, C, D)$ iff complete(of A) or complete(of B) or complete(of C) or complete(of D)
--------------------	---

Die Regel *succession* setzt sich aus anderen Regeln zusammen und muss daher nicht gesondert erläutert werden. Alle anderen Regelvorlagen sind elementar. Ihre Auswirkungen sollen anhand des folgenden DPIL-Modells illustriert werden:

```

process Declare {
  task A
  task B
  task C
  task D

  advise "existence(A)": existence(A)
  advise "init(A)": init(A)
  advise "response(A, B)": response(A, B)
  advise "not co-existence(A, D)": notCoExistence(A, D)
  advise "1 of(A, B, C, D)": _1of4(A, B, C, D)
}

```

Abbildung 7.2 zeigt die Aufgabensicht während der Ausführung des exemplarischen Modells. Ansicht ❶ zeigt die Aufgabenliste unmittelbar nach dem Start des Prozesses. Nur Aufgabe ist neutral bewertet (blau). Aufgabe B ist nicht empfohlen (orange), weil ihr Starten gegen die Regel *init(A)* und *precedence(A, B)* verstoßen würde: der Prozess sollte mit A beginnen und vor B sollte A durchgeführt werden. Ebenso würden die Aufgaben C und D gegen in *init(A)* verstoßen. Unterhalb der Aufgabenliste werden die derzeit verletzten Regeln angezeigt. Nach dem Start des Prozesses sind die Regeln *existence(A)* und *1 of(A, B, C, D)* verletzt, weil zu diesem Zeitpunkt noch keine der Aufgaben durchgeführt wurde.

Ansicht ❷ zeigt die Aufgabenliste nach der Durchführung von Aufgabe A. Aufgabe B ist zu diesem Zeitpunkt empfohlen, weil ihr Starten die Regel *response(A, B)* erfüllen würde. Aufgabe D würde die Regel *not co-existence(A, D)* verletzen, wonach A und D nicht beide durchgeführt werden sollten. Nachdem A ausgeführt wurde, sind die Regeln *existence(A)* und *1 of(A, B, C, D)* nicht weiter verletzt. Im Gegenzug ist nun aber die Regel *response(A, B)* verletzt, wonach auf A irgendwann B folgen sollte.

Ansicht ❸ zeigt die Aufgabensicht nach dem Abschließen von Aufgabe B. Lediglich Aufgabe D ist weiterhin nicht empfohlen, weil A und D nicht im selben Prozess ausgeführt werden sollten. Alle anderen Aufgaben sind neutral markiert und die Prozessinstanz verstößt gegen keine der Regeln mehr.

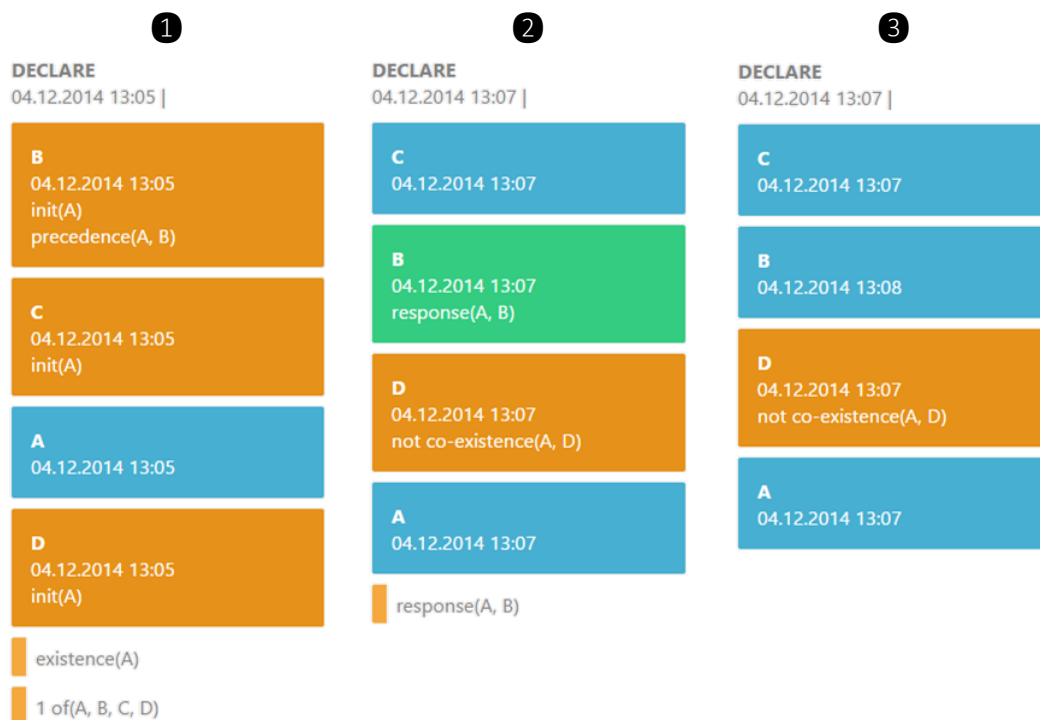


Abbildung 7.2 Aufgabsicht bei der Ausführung des DECLARE-Beispiels nach dem Start, nach A und nach B

Das Beispiel zeigt, dass DECLARE und DPIL zwar unterschiedlichen Prinzipien folgen, sich das Verhalten von DECLARE aber mit DPIL und der PN-Plattform problemlos nachbilden lässt.

7.8 Abbildung von CMMN auf DPIL

DPIL stellt die Sprache des regelbasierten Anteils der PN-Plattform dar. Als rein textuelle Modellierungssprache eignet sie sich jedoch nur bedingt für eine intuitive Prozessmodellierung und die Kommunikation zwischen Geschäfts- und IT-Abteilungen. Eine grafische Repräsentation aller Elemente von DPIL ist nicht Teil dieser Arbeit. Mit der *Case Management Model and Notation* (CMMN) existiert allerdings eine aktuelle regelbasierte Prozessmodellierungssprache, die sich auf DPIL abbilden lässt. Sie kann nicht zwischen verschiedenen Modalitäten unterscheiden und erlaubt eine perspektivenübergreifende Modellierung nur bedingt. Es finden sich also nicht alle von DPIL unterstützten Konzepte auch in CMMN wieder. Dennoch stellt CMMN den aktuellsten Ansatz zur grafischen regelbasierten Prozessmodellierung dar. Im Folgenden wird daher beschrieben, wie sich CMMN-Prozessmodelle auf DPIL-Modelle abbilden und somit auf der PN-Plattform ausführen lassen.

7.8.1 Strukturelle Bestandteile

Abbildung 7.3 zeigt die strukturellen Bestandteile eines CMMN-Modells. Ein Fall (Case) setzt sich aus mindestens einem Abschnitt (Stage) zusammen, der wiederum in Abschnitte untergliedert werden kann. Elementare Bestandteile eines Abschnitts können personelle Aufgaben (HumanTask), Meilensteine (Milestone), Prozesse (ProcessTask) oder wiederum Fälle (CaseTask) sein. Eine personelle Aufgabe wird durch einen Fallteilnehmer durchgeführt, der die entsprechende Rolle (Role) einnimmt. Ein Meilenstein stellt ein erreichbares Ziel innerhalb des Prozesses dar. Er wird vor allem definiert um den Fortschritt des Falls bestimmen zu können. Ein

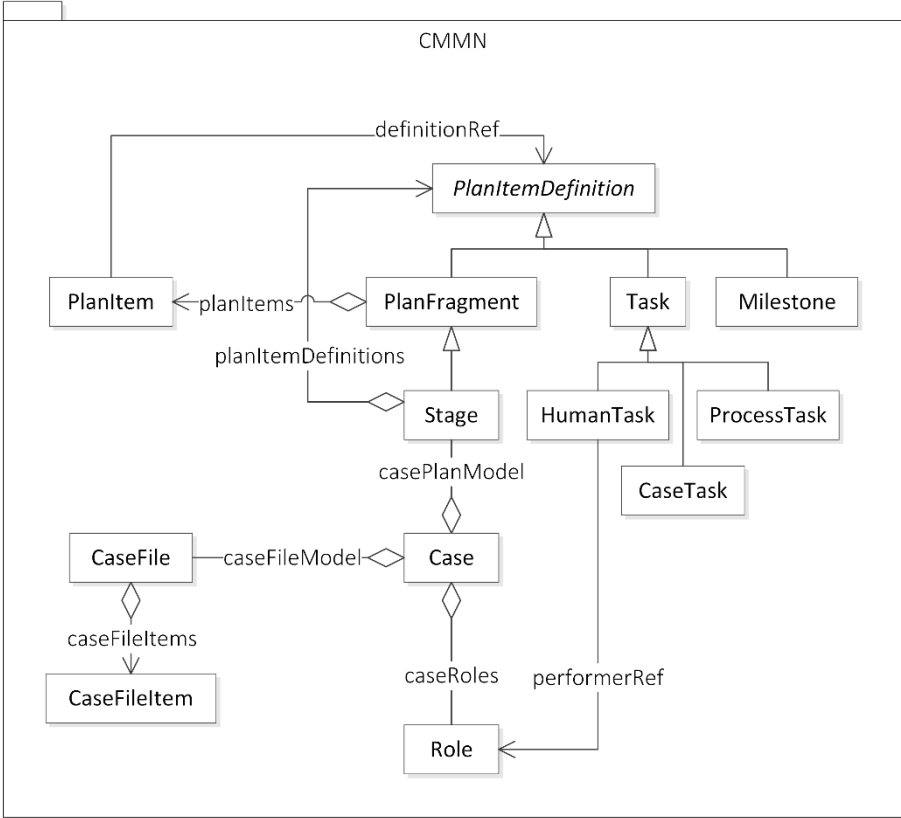


Abbildung 7.3 Strukturelle Bestandteile eines CMMN-Prozesses

7.8.2 Prozessregeln

sich auf das Ereignis eines Elements der Fallakte (**CaseFileItemOnPart**) oder auf ein Ereignis eines Planelements (**PlanItemOnPart**) beziehen.

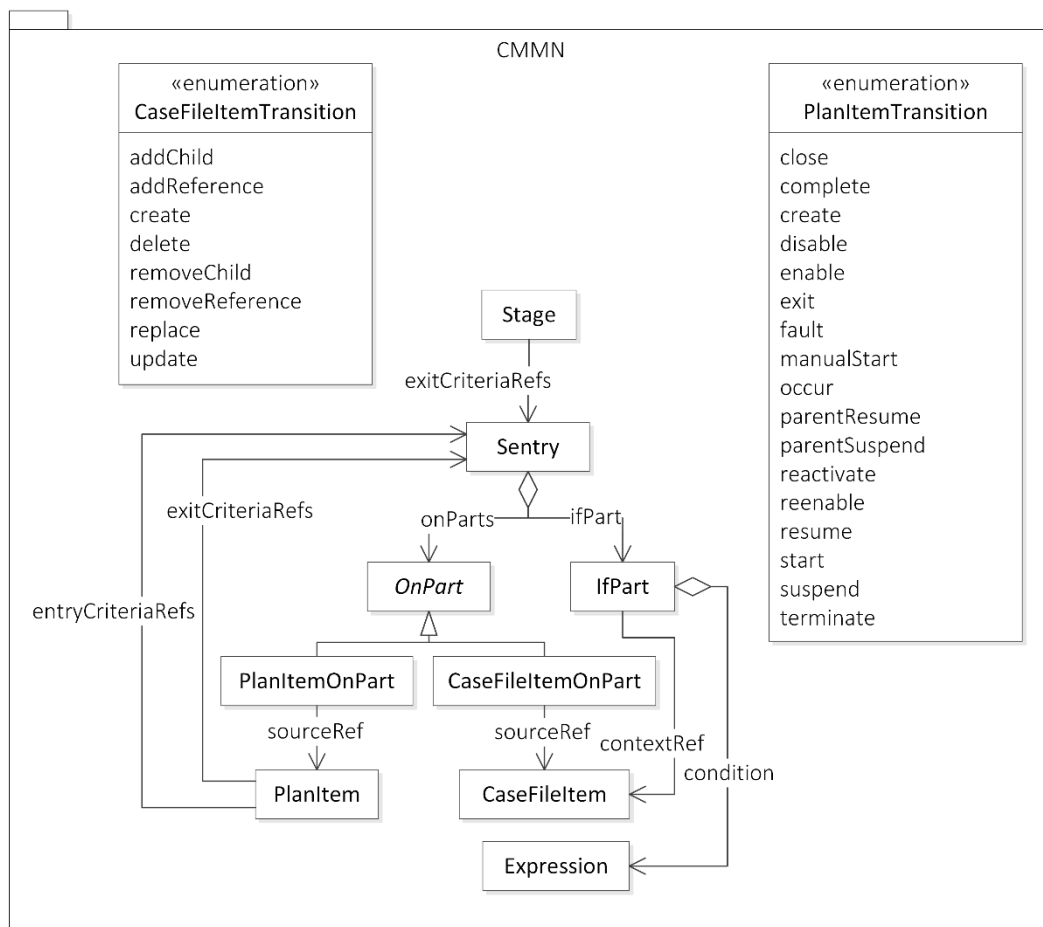


Abbildung 7.4 Bestandteile von CMMN zur Modellierung von Regeln

CMMN greift also auf ähnliche Konzepte zurück wie BPMN. Es umfasst Prozesse, die aus Aktivitäten, Datenobjekten oder wiederum Prozessen bestehen. Statt das Verhalten des Prozesses jedoch mittels Ereignissen, Gattern und Kontrollflüssen zu beschreiben, werden die Aktivitäten des Modells auf Ereignisse von anderen Aktivitäten oder Datenobjekten hin aktiviert.

7.8.3 Abbildung auf DPIL

Die Kernelemente, also Aufgaben, Meilensteine, Abschnitte und Wachen, lassen sich auf entsprechende Konzepte in DPIL abbilden. Das resultierende DPIL-Modell lässt sich wiederum auf der PN-Plattform ausführen. Im Folgenden wird für jedes der CMMN-Konzepte beschrieben, wie es auf DPIL abgebildet werden kann.

Jedes Element der Fallakte in CMMN entspricht einem Datenobjekt in DPIL. Die Fallakte ist an den jeweiligen Fall gebunden, gilt also für alle Abschnitte innerhalb des Falls. Die entsprechenden Datenobjekte müssen also auf der Ebene des obersten Prozesses definiert werden, damit ihr Geltungsbereich dem der Elemente der Fallakte entspricht. DPIL unterstützt zwei Arten von Datenobjekten: Dokumente in einem externen ECM-System und interne Variablen.

Dementsprechend wird ein Element der Fallakte vom Typ `CMISDocument` in ein DPIL-Dokument übersetzt und ein Element vom Typ `String` exemplarisch in eine Zeichenkettenvariable.

DPIL kennt keine Unterscheidung zwischen der Vorlage und der Ausprägung eines Modellelements, also zwischen `PlanItemDefinition` und `PlanItem`. Vorlagen und Ausprägung werden jeweils zusammengelegt.

Die Eingangs- und Ausgangskriterien für ein Planelement werden in entsprechende Regeln übersetzt. Eine Wache kann auf mehrere Ereignisse (`OnPart`) reagieren, von denen alle eintreten und die Bedingung (`IfPart`) – falls eine angegeben wurde – zutreffen muss, damit der entsprechende Übergang ausgelöst wird (konjunktive Semantik, UND). Bei mehreren Wachen muss nur eine zutreffen (disjunktive Semantik, ODER). Ein erfülltes Eingangskriterium setzt einen Abschnitt oder eine Aufgabe von verfügbar (engl. *available*) nach freigegeben (engl. *enabled*) oder aktiv (engl. *active*) und einen Meilenstein auf erfüllt (engl. *achieved*). Ein erfülltes Ausgangskriterium setzt einen Abschnitt oder eine Aufgabe von aktiv nach beendet (engl. *terminated*), was zum Ausgangsereignis (engl. *exit*) führt. Die Ausgangskriterien eines Abschnitts in CMMN entsprechen den Meilensteinen eines Prozesses in DPIL.

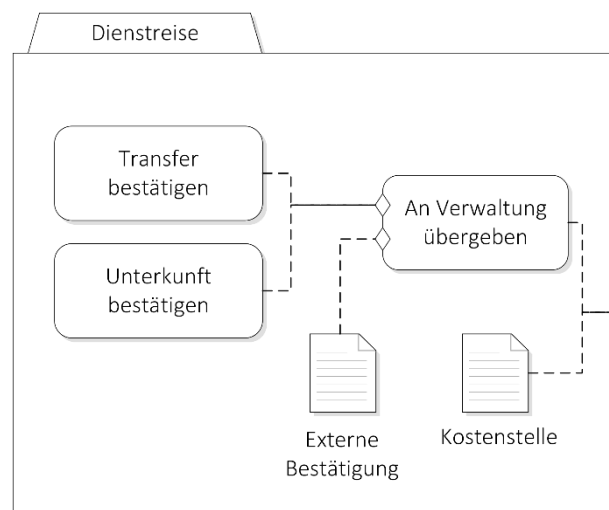


Abbildung 7.5 Exemplarisches CMMN-Modell zur Überführung in DPIL

Das ist Abbildung 7.5 gezeigte Modell beschreibt einen Ausschnitt aus einem Dienstreisevorgang. Zum Beginnen der Aktivität „An Verwaltung übergeben“ ist der Abschluss von „Transfer bestätigen“ und „Unterkunft bestätigen“ oder das Vorliegen von „Externe Bestätigung“ notwendig. Um den Prozess abzuschließen, muss wiederum „An Verwaltung übergeben“ abgeschlossen sein und „Kostenstelle“ vorliegen. Dieses Modell lässt gemäß den oben beschriebenen Regeln in folgendes DPIL-Modell übersetzen:

```

process DR "Dienstreise" {
  task T "Transfer bestätigen"
  task U "Unterkunft bestätigen"
  task V "An Verwaltung übergeben"

```

```
variable E "Externe Bestätigung"  
variable K "Kostenstelle"  
  
ensure start(of V) implies (complete(of T) and complete(of U)) or  
write(of E)  
  
milestone complete(of V) and write(of K)  
}
```

Die Eingangsbedingungen für „An Verwaltung übergeben“ wurden in eine entsprechende harte Regel (**ensure**) übersetzt, die Ausgangsbedingung für den gesamten Prozess in einen entsprechenden Meilenstein (**milestone**).

7.9 Zusammenfassung und Bewertung

Um zu zeigen, dass sich die Sprache DPIL zur Modellierung agiler Prozesse eignet, wird sie anhand einer Sammlung von häufigen Mustern des Prozessmanagements evaluiert. Es wird die Abdeckung von Mustern aus den Bereichen Kontrollfluss, Daten und Organisation untersucht und mit der Abdeckung von BPMN verglichen.

Die Evaluation zeigt, dass DPIL der Sprache BPMN in den Bereichen Daten und Organisation überlegen ist. Im Bereich des Kontrollflusses ist BPMN hingegen DPIL überlegen. Es lassen sich also mehr Muster strikter Abläufe mit BPMN modellieren. DPIL ist hingegen besser geeignet, um übergreifender Muster aus der datenbezogenen und organisatorischen Perspektive darzustellen. Beide Modellierungsprinzipien decken also den ursprünglich vermuteten Bereich von Prozessen ab. Darüber hinaus zeigt sich, dass DPIL insgesamt mehr Muster als BPMN abdeckt und damit insgesamt besser zur Modellierung von Prozessen geeignet ist.

Darüber hinaus lassen sich die zentralen sprachlichen Konzepte sowohl der im wissenschaftlichen Umfeld verbreiteten DECLARE-Plattform als auch der CMMN auf DPIL abbilden. Dabei bleibt die Semantik der jeweiligen Modelle erhalten, so dass sich sowohl DECLARE- als auch CMMN-Modelle prinzipiell auf der PN-Plattform ausführen lassen.

8 Zusammenfassung und zukünftige Aufgabenfelder

Ausgangspunkt für die vorliegende Arbeit bildet die Beobachtung, dass sich die Prozesslandschaft einer Organisation aus Prozessen verschiedener Natur zusammensetzt. Während das bestehende Prozessmanagement die eher strikten Prozesse adressiert, werden die eher agilen Prozesse nur unzureichend unterstützt. Es wird gezeigt, dass zur Modellierung agiler Prozesse eine übergreifende Betrachtung aller grundlegenden Perspektiven notwendig ist, dass verpflichtende und empfohlene Handlungen unterschieden werden müssen und dass die Empfehlungen des Ausführungssystems dem Prozessteilnehmer erklärt werden müssen. Diese agilen Prozesse, deren Abläufe sich nicht bereits im Voraus weitgehend spezifizieren lassen, können durch bestehende Lösungen weder adäquat modelliert, noch IT-unterstützt ausgeführt werden. Im Rahmen der Arbeit wird daher ein Prozessausführungssystem entwickelt, das in der Lage ist, Prozesse aus strikten und agilen Anteilen integriert auszuführen und dabei die besonderen Anforderungen agiler Prozesse berücksichtigt.

Strikte Teilprozesse können in der standardisierten Notation BPMN modelliert werden. Zu ihrer Ausführung wurde das Konzept der *Process Virtual Machine* nach Baeyens und Faura adaptiert.

Zur Modellierung agiler Prozesse wird im Rahmen der Arbeit die textuelle Sprache DPIL entwickelt, die eine übergreifende Behandlung aller grundlegenden Perspektiven und eine Unterscheidung in verpflichtende und empfohlene Handlungen erlaubt. Zu ihrer Ausführung wird eine virtuelle Maschine entworfen, die in der Lage ist, empfohlene Handlungen auf das Prozessmodell zurückzuführen.

Zur Modellierung agiler Prozesse eignet sich zumindest zum Teil auch die standardisierte Notation CMMN. Sie erlaubt eine übergreifende Modellierung aller grundlegenden Perspektiven nur bedingt und ermöglicht keine Unterscheidung von verpflichtenden und empfohlenen Handlungen. Dennoch verfügt CMMN über grafische Repräsentation und ist daher besser zur Vermittlung zwischen IT und Fachabteilungen geeignet. Um sowohl für strikte als auch für agile Prozesse eine grafische Modellierung zu unterstützen, können CMMN-Modelle auf DPIL-Modelle abgebildet und auf dem System ausgeführt werden.

Im Folgenden werden fünf Bereiche identifiziert, in denen das vorgestellte Konzept im Rahmen von zukünftigen Arbeiten verbessert oder erweitert werden kann.

8.1 Migration von Prozessinstanzen auf geänderte Prozessmodelle

Wird ein Prozessmodell geändert, so müssen alle laufenden Instanzen des Modells entweder im alten Modell abgeschlossen, abgebrochen und im neuen Modell erneut begonnen oder auf das neue Modell übertragen (migriert) werden. Bereits in (M Pešić et al. 2007) wird gezeigt, dass sich laufende Instanzen regelbasierter Prozessmodelle leichter in ein geändertes Modell migrieren lassen, als es für prozedurale Modelle der Fall ist. Grund hierfür ist, dass im regelbasierten Fall kein passender Zustand im neuen Modell gefunden werden muss. Wenn der bisherige Verlauf im neuen Modell eine harte Regel verletzen würde, ist die Migration nicht möglich. In jedem anderen

Fall kann der bisherige Verlauf übernommen und die Ausführung ohne zusätzlichen Aufwand nach dem neuen Modell fortgeführt werden. Eine Migration von Prozessinstanzen wird durch das Konzept der PN-Plattform nicht ausgeschlossen, ist aber nicht Teil der vorliegenden Arbeit.

8.2 Modellierung und Ausführung von mehrfach instanzierbaren Aktivitäten

In (van der Aalst et al. 2003) werden verschiedene Szenarien für Aktivitäten aufgeführt, die mehrmals innerhalb einer Prozessinstanz nebenläufig zueinander gestartet werden können. Auch die BPMN sieht komplexe Konstrukte (`MultiInstanceLoopCharacteristics`) zur Abbildung solcher Aktivitäten vor (Object Management Group Inc. 2011). Die RPVM als Grundlage der Ausführung regelbasierter Modelle verwaltet keinen Zustand für die Instanzen von Aktivitäten. Sie liefert lediglich die Bewertung der Ereignisse einer Sprache. Um mehrfach instanzierbare Aktivitäten abzudecken, müssten die Sprache DPIL und ihre Anbindung an die RPVM erweitert werden. Im Prozessmodell müssen die Aktivitäten entsprechend markiert werden. Außerdem müssen zu einer Aktivität im Modell eventuell mehrere Zustände verwaltet und mit der Aufgabenverwaltung koordiniert werden.

8.3 Grafische Darstellung regelbasierter Prozessmodelle

Im Rahmen dieser Arbeit wird eine textuelle regelbasierte Zwischensprache zur Prozessmodellierung beschrieben, die die gestellten Anforderungen erfüllt. Das Ziel künftiger Arbeiten muss es sein, eine geeignete grafische Repräsentation dieser Sprache zu finden. Die CMMN wird in dieser Arbeit bereits berücksichtigt und bildet einen Ausgangspunkt.

CMMN erlaubt derzeit nur eine Verknüpfung von Aufgaben mit einer Rolle und sonst keine weiteren Konzepte zur Unterstützung der organisatorischen Perspektive. Das Konzept der Pools und Lanes aus BPMN lässt sich auf CMMN übertragen. Diese Rechtecke bieten eine einfache Möglichkeit, um Zuständigkeiten darzustellen. Im Modell dieser Elemente lassen sich dann genauere Informationen zu Gruppen, Rollen, Fähigkeiten etc. transportieren. Außerdem lassen sich grafische Elemente nutzen, um die Regeln für eine Bindung oder Trennung von Zuständigkeiten darzustellen. Ausgefüllte Ellipsen könnten zum Beispiel Aktivitäten markieren, die alle vom selben Prozessteilnehmer durchgeführt werden sollen. Nicht ausgefüllte Ellipsen könnten die markieren, die von verschiedenen Teilnehmern durchgeführt werden müssen.

Eine Unterscheidung von verpflichtenden und empfohlenen Handlungen ist in CMMN derzeit nicht möglich. Regeln werden in CMMN durch Wachen (engl. *sentries*) realisiert. Eine Möglichkeit, die Unterscheidung auf einfache Art und Weise nachzureichen besteht in der entsprechenden Markierung dieser Wachen. Eine Wache verbindet immer Aufgaben oder Informationseinheiten. Die entsprechende Linie könnte durchgezogen für eine harte Regel und gestrichelt für eine weiche Regel gezeichnet werden.

Der in Abschnitt 1.3.6 gezeigte Aufnahmeprozess einer psychiatrischen Klinik ließe sich dann wie in Abbildung 8.1 mit Hilfe von BPMN und der erweiterten Variante von CMMN modellieren. Die ursprünglich als Anmerkung hinterlegten Empfehlungen in den Teilprozessen ① und ④ ließen sich dann als Regeln ausdrücken. Die Datenobjekte in den Teilprozessen müssen in jedem Fall erzeugt werden um die entsprechenden Aufgaben abschließen zu können. Die ausgefüllten Diamantsymbole an den Aufgaben, also die Ausgangsbedingungen, sind daher mit

durchgezogenen Linien mit den Datenobjekten verbunden (②). Die Reihenfolge der Aufgaben ist eine Empfehlung aber nicht verpflichtend. Die nicht ausgefüllten Diamantsymbole der jeweils folgenden Aufgaben, also deren Eingangsbedingung sind daher mit gestrichelten Linien mit den vorangehenden Aufgaben verbunden (③).

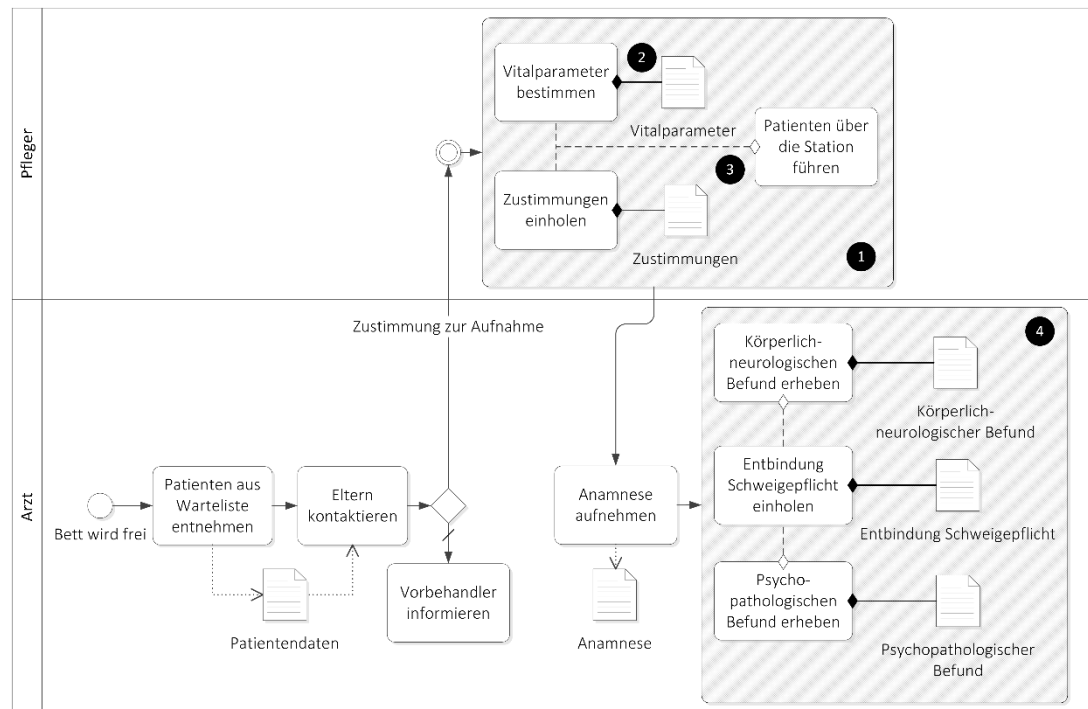


Abbildung 8.1 Hybride Modellierung des Aufnahmeprozesses einer psychiatrischen Klinik

Die Reihenfolgen sind damit zunächst beliebig. Ein Prozessteilnehmer würde aber darauf hingewiesen werden, dass in Teilprozess ① der Aufgabe „Patienten über die Station führen“ die beiden anderen Aufgaben vorangehen sollten und dass in Teilprozess ④ zunächst die Entbindung von der Schweigepflicht eingeholt werden sollte, bevor die beiden Befunde erhoben werden. Neben diesen Aspekten, die die Verhaltens- und Informationsperspektive betreffen, kann auch die Organisationsperspektive berücksichtigt werden. Die beiden agilen Teilprozesse sind unterschiedlichen Lanes zugeordnet, dem Arzt und dem Pfleger. Wie auch die Unterscheidung in verpflichtende und empfohlene Regeln (durchgezogene und gestrichelte Linien) ist auch die Platzierung in Lanes nicht in der offiziellen Version von CMMN vorgesehen und müsste erst spezifiziert werden.

8.4 Prozedurale und regelbasierte Modellierung in einer Sprache

Eingangs wird gezeigt, dass sich eine prozedurale Modellierung eher für strikte und eine regelbasierte Modellierung eher für agile Prozesse eignet. Aufgrund der unterschiedlichen Prinzipien sowohl während der Modellierung, als auch während der Interpretation und Ausführung wird stets angenommen, dass es sich um verschiedene Sprachen handeln muss. Dies zeigt sich auch in den grundsätzlich verschiedenen Sprachen BPMN und CMMN.

Würde man prozedurale Sprachen wie BPMN auf DPIL abbilden können, dann könnte man prozedurale und regelbasierte Modelle auf derselben virtuellen Maschine ausführen. Eine Möglichkeit könnte darin bestehen, den Petri-Netz-Formalismus in DPIL nachzubilden (Lam 2012; Wong & Gibbons 2007). Ließen sich prozedurale und regelbasierte Modelle auf einer Engine ausführen, dann wäre damit auch der Grundstein für eine Zusammenführung der beiden Modellierungsprinzipien gelegt (Westergaard & Slaats 2013).

8.5 Vorausschau und Simulation von Ereignisketten

Die regelbasierte Prozess-VM (RPVM) führt effektiv eine Simulation von jeweils einem Zeitschritt in die Zukunft durch. Eine weitergehende Simulation ist für verschiedene Anwendungsfälle interessant:

- Die Empfehlung an die Prozessteilnehmer gilt jeweils nur für den folgenden Zeitschritt und führt daher zu einer lokalen Optimierung des Prozessverlaufs. Eine Handlung, die im aktuellen Zeitschritt empfohlen wird kann dazu führen, dass in einem späteren Zeitschritt bestimmte Handlungen nicht mehr möglich sind, also ein bestimmter Zustand nicht mehr erreicht werden kann. In manchen Anwendungsgebieten kann dieses Verhalten zu Problemen führen. Eine weitergehende Simulation würde zu einer globaleren Optimierung des Prozessverlaufs führen und „Sackgassen“ lassen sich vermeiden.
- Die regelbasierte Modellierung von Prozessen gestaltet sich schwieriger als die prozedurale, weil die durch das Modell abgebildeten Abläufe nicht mehr direkt erkennbar sind. Während des Entwurfs könnten Teile des Modells unter bestimmten Annahmen simuliert und aus den entstehenden Ereignisverläufen prozedurale Modelle rekonstruiert werden (engl. *process mining*). Die prozeduralen „Sichten“ auf das Modell erleichtern sein Verständnis.

Um mehr als ein Ereignis in die Zukunft zu sehen, müsste die RPVM zumindest theoretisch alle möglichen Ereignisketten bis zu einem gewissen Zielzustand simulieren. Um den Aufwand zu verringern, könnte man nur bis zu bestimmten Meilensteinen simulieren. Die Schwierigkeit besteht vor allem im Umgang mit Datenobjekten, deren Werte ebenfalls simuliert werden müssten.

Quellenverzeichnis

- Van der Aalst, W.M.P., 1998. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1), pp.21–66.
- Van der Aalst, W.M.P., 1996. Three Good Reasons for Using a Petri-net-based Workflow Management System. In *International Working Conference on Information and Process Integration in Enterprises (IPIC '96)*. Cambridge, MA, USA: Springer, pp. 179–201.
- Van der Aalst, W.M.P., 2009. TomTom for Business Process Management (TomTom4BPM). In *21st International Conference on Advanced Information Systems Engineering (CAiSE 2009)*. Amsterdam, NL: Springer, pp. 2–5.
- Van der Aalst, W.M.P. et al., 2003. Workflow Patterns. *Distributed and Parallel Databases*, 14(3), pp.5–51.
- Van der Aalst, W.M.P., Weske, M. & Grünbauer, D., 2005. Case Handling : A New Paradigm for Business Process Support. *Data & Knowledge Engineering*, 53(2), pp.129–162.
- Baeyens, T. & Faura, M.V., 2007. The Process Virtual Machine. Available at: <http://docs.jboss.com/jbpm/pvm/article/>.
- Bartonitz, M., 2009. Historische Entwicklung wichtiger Standards im Business Process Management. Available at: http://commons.wikimedia.org/wiki/File:Standards_BPM_2009_11.jpg.
- Bertoa, M.F. & Vallecillo, A., 2010. Quality Attributes for Software Metamodels. In *13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAQOSE 2010)*. Málaga, ES.
- Bußler, C., 1998. *Organisationsverwaltung in Workflow-Management-Systemen*, Wiesbaden: DUV.
- Le Clair, C. & Miers, D., 2011. The Forrester Wave™ : Dynamic Case Management, Q1 2011.
- Clark, T., Sammut, P. & Willans, J., 2008. *Applied Metamodelling: A Foundation for Language Driven Development* 2nd ed., Ceteva.
- CMMI Product Team, 2011. *CMMI für Entwicklung, Version 1.3*, Pittsburgh, PA, US.
- Von der Crone, H.C. & Roth, K., 2003. Der Sarbanes-Oxley Act und seine extraterritoriale Bedeutung. *Aktuelle Juristische Praxis*, (2), pp.131–140.
- Curtis, B., 1992. Process Modeling. *Communications of the ACM*, 35(9), pp.75–90.
- Decker, G., Tscheschner, W. & Puchan, J., 2009. Migration von EPK zu BPMN. In M. Nüttgens et al., eds. *8. Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises „Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)“*. Berlin: Gesellschaft für Informatik, pp. 91–109.
- Deming, W.E., 1982. *Out of the Crisis*, Cambridge, MA, USA: Massachusetts Institute of Technology.
- Dijkstra, E.W., 1982. On the Role of Scientific Thought. In *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, pp. 60–66.
- Dourish, P. et al., 1996. Freeflow: Mediating Between Representation and Action in Workflow Systems. In *ACM Conference on Computer Supported Cooperative Work (CSCW '96)*. Boston, MA, US: ACM.
- Eclipse Foundation, I., 2014. Xtext - Language Development Made Easy! Available at: <http://www.eclipse.org/Xtext/>.
- European Computer Manufacturers Association (ECMA), 1966. Standard ECMA-4 - Flow Charts (2nd Edition, withdrawn).
- Fachnormenausschuß Informationsverarbeitung (FNI) im Deutschen Normenausschuß (DNA), 1966. DIN 66 001 - Sinnbilder für Datenfluß- und Programmablaufpläne.
- Fahland, D. et al., 2009. Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. In *Enterprise, Business-Process and Information Systems Modeling (10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009)*. Amsterdam, The Netherlands: Springer, pp. 353–366.
- Forgy, C.L., 1982. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1), pp.17–37.
- Fowler, M., 2011. *Domain-Specific Languages*, Addison-Wesley.

- Fowler, M., 2002. *Patterns of Enterprise Application Architecture*, Boston, MA, US: Addison-Wesley Longman Publishing.
- Gamma, E. et al., 1994. *Design Patterns - Elements of Reusable Object-Oriented Software*, Prentice Hall.
- Gfeller, B., Völzer, H. & Wilmsmann, G., 2011. Faster Or-join Enactment for BPMN 2.0. In *Lecture Notes in Business Information Processing*. Springer Berlin Heidelberg, pp. 31–43.
- Gilbert, M.R. et al., 2013. *Magic Quadrant for Enterprise Content Management*.
- Glance, N.S., Pagani, D.S. & Pareschi, R., 1996. Generalized Process Structure Grammars (GPSG) for Flexible Representations of Work.
- Goedertier, S. & Vanthienen, J., 2007. Declarative Process Modeling with Business Vocabulary and Business Rules. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*. Springer, pp. 603–612.
- Goldschmidt, T., Becker, S. & Uhl, A., 2008. Classification of Concrete Textual Syntax Mapping Approaches. In I. Schieferdecker & A. Hartman, eds. *4th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2008)*. Berlin, DE: Springer Berlin / Heidelberg, pp. 169–184.
- Gronau, N. & Weber, E., 2004. Management of Knowledge Intensive Business Processes. In *Business Process Management*. pp. 163–178.
- Günther, C.W., 2009. OpenXES Developer Guide. Available at: <http://www.xes-standard.org/openxes/developerguide>.
- Hammer, M. & Champy, J., 1993. *Reengineering the Corporation: A Manifesto for Business Revolution* 3rd ed., HarperBusiness.
- Hevner, A.R. et al., 2004. Design Science in Information Systems Research. *MIS Quarterly*, 28(1), pp.75–105.
- Hoheisel, A., Dollmann, T. & Fellmann, M., 2009. Überführung von EPK-Modellen in ausführbare Grid- und Cloud-Prozesse. In M. Nüttgens et al., eds. *Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK 2009)*. Bonn: Gesellschaft für Informatik e.V. (GI), pp. 118–137.
- Igler, M., 2011. *ESProNa - Eine Constraintsprache zur multimodalen Prozessmodellierung und navigationsgestützten Ausführung*. Universität Bayreuth.
- Igler, M., Jablonski, S., et al., 2010. ESProNa: Constraint-based Declarative Business Process Modeling. In *14th IEEE International Enterprise Distributed Object Computing Conference (EDOCW 2010)*. Vitória, ES, BR: IEEE Computer Society, pp. 91–98.
- Igler, M., Faerber, M., et al., 2010. Modeling and Planning Collaboration in Process Management Systems using Organizational Constraints. In *6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2010)*. Chicago, IL, USA: IEEE, pp. 1–10.
- Jablonski, S., 1994. MOBILE: A Modular Workflow Model and Architecture. In A. Verbraeck, H. G. Sol, & P. W. G. Bots, eds. *4th International Working Conference on Dynamic Modelling and Information Systems*. Noordwijkerhout, NL: Delft University Press, pp. 1–30.
- Jablonski, S. & Bußler, C., 1996. *Workflow Management: Modeling Concepts, Architecture and Implementation*, London: Thomson.
- JBoss Drools Team, 2013. JBoss Drools Documentation - Chapter 7: Rule Language Reference. Available at: <http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/DroolsLanguageReferenceChapter.html>.
- Josuttis, N., 2008. *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*, dpunkt.
- Kloppmann, M. et al., 2005. *WS-BPEL Extension for Sub-processes: BPEL-SPE*, Available at: <http://xml.coverpages.org/BPEL-SPE-Subprocesses.pdf>.
- Lam, V.S.W., 2012. A Precise Execution Semantics for BPMN. *IAENG International Journal of Computer Science*, 39(1).
- Lohse, G.L., 1993. A Cognitive Model for Understanding Graphical Perception. *Human-Computer Interaction*, 8, pp.353–388.
- Luftman, J., 2005. Key Issues for IT Executives 2004. *MIS Quarterly Executive*, 4(2), pp.269–286.

- Mackinlay, J., 1986. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics*, 5(2), pp.110–141.
- Meerkamm, S., 2011. *Ein Rahmenwerk für das Prozessdesign zur Identifikation, Klassifikation und Umsetzung von Anforderungen*. Universität Bayreuth.
- Montali, M. et al., 2009. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, V(May).
- Zur Muehlen, M. & Ho, D.T., 2006. Risk Management in the BPM Lifecycle. In *BPM 2005 Workshops*. Springer-Verlag Berlin Heidelberg, pp. 454–466.
- Zur Muehlen, M., Indulska, M. & Kittel, K., 2008. Towards Integrated Modeling of Business Processes and Business Rules. In *19th Australasian Conference on Information Systems*.
- Zur Muehlen, M. & Recker, J., 2008. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In pp. 465–479.
- Müller, F., Brown, J. & Potts, J., 2013. *CMIS and Apache Chemistry in Action*, Manning Publications.
- Nonnengart, A. & Ohlbach, H.J., 1992. Modal- und Temporallogik. In K. H. Bläsius & H.-J. Bürckert, eds. *Deduktionssysteme: Automatisierung des logischen Denkens*. Oldenbourg, pp. 239–284.
- OASIS, 2012. Content Management Interoperability Services (CMIS) Version 1.1. Available at: <http://docs.oasis-open.org/cmisis/CMIS/v1.1/CMIS-v1.1.html>.
- OASIS, 2010. Web Services – Human Task (WS-HumanTask) Specification Version 1.1. Available at: <http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cs-01.html>.
- OASIS, 2007a. Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Available at: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>.
- OASIS, 2007b. WS-BPEL Extension for People (People4People), Version 1.0. Available at: <http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html>.
- Object Management Group, 2014a. Case Management Model and Notation (CMMN) - Version 1.0. Available at: <http://www.omg.org/spec/CMMN/1.0/PDF/>.
- Object Management Group, 2014b. Decision Model and Notation - Beta 1. Available at: <http://www.omg.org/spec/DMN/1.0/Beta1/PDF/>.
- Object Management Group, 2013. Semantics of Business Vocabulary and Business Rules (SBVR) - Version 1.2. Available at: <http://www.omg.org/spec/SBVR/1.2/>.
- Object Management Group Inc., 2011. Business Process Model and Notation (BPMN) Version 2.0. Available at: <http://www.omg.org/spec/BPMN/2.0>.
- Oracle Corporation, 2013. JSR-341 Expression Language Specification Version 3.0 Final Release.
- Pareto, V. & Schiavini, A.S., 1971. *Manual of Political Economy*, Augustus M. Kelley.
- Pešić, M., 2006. *Constraint-Based Workflow Management Systems: Shifting Control to Users*. Technische Universiteit Eindhoven.
- Pešić, M. et al., 2007. Constraint-Based Workflow Models: Change Made Easy. In R. Meersman & Z. Tari, eds. *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, IS*. Springer Berlin / Heidelberg, pp. 77–94. Available at: http://dx.doi.org/10.1007/978-3-540-76848-7_7.
- Pešić, M., Schonenberg, H. & van der Aalst, W.M.P., 2007. DECLARE: Full Support for Loosely-Structured Processes. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. Annapolis, Maryland, USA, p. 287.
- PKWARE Inc., 2012. .ZIP File Format Specification Version 6.3.3. Available at: <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>.
- Regev, G. & Wegmann, A., 2005. A Regulation-Based View on Business Process and Supporting System Flexibility. In *17th International Conference on Advanced Information Systems Engineering (CAiSE '05) Workshops*. Porto, pp. 35–42.
- Reichert, M. & Weber, B., 2012. *Enabling Flexibility in Process-Aware Information Systems*, Springer-Verlag Berlin Heidelberg.
- Reijers, H.A., Slaats, T. & Stahl, C., 2013. Declarative Modeling — An Academic Dream or the Future for BPM? In *11th International Conference on Business Process Management (BPM 2013)*. Beijing, China: Springer Berlin Heidelberg, pp. 307–322.

- De Roover, W., Caron, F. & Vanthienen, J., 2012. A Prototype Tool for the Event-Driven Enforcement of SBVR Business Rules. In *Business Process Management Workshops, BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*. Springer Berlin Heidelberg, pp. 446–457.
- Russell, N., Hofstede, A.H.M. ter, et al., 2004. *Workflow Data Patterns*, Brisbane: Queensland University of Technology.
- Russell, N., ter Hofstede, A.H.M., et al., 2004. *Workflow Resource Patterns*, Eindhoven: Eindhoven University of Technology.
- Sadiq, S., Sadiq, W. & Orłowska, M., 2001. Pockets of Flexibility in Workflow Specification. In *20th International Conference on Conceptual Modeling (ER'2001)*. Yokohama, Japan: Springer, pp. 513–526.
- Scheer, A.-W., Thomas, O. & Adam, O., 2005. Process Modeling Using Event-Driven Process Chains. In M. Dumas, W. M. P. van der Aalst, & A. H. M. Hofstede ter, eds. *Process-Aware Information Systems*. Hoboken, New Jersey: John Wiley & Sons, Inc., pp. 119–145.
- Schlundt, M., 2004. *Historienverwaltung in Workflow-Management-Systemen*. Friedrich-Alexander-Universität Erlangen-Nürnberg.
- Schonenberg, H. et al., 2008. Process Flexibility: a Survey of Contemporary Approaches. In *Advances in Enterprise Engineering I, 4th International Workshop CIAO! and 4th International Workshop EOMAS, CAiSE 2008*. Montpellier, France: Springer, pp. 16–30.
- Smith, J.E. & Ravi, N., 2005. *Virtual Machines: Versatile Platforms for Systems and Processes*, Elsevier B.V.
- Software AG, 2012. ARIS Method - ARIS Platform Version 7.2 - Service Release 3.
- Stiehl, V., 2013. *Prozessgesteuerte Anwendungen entwickeln und ausführen mit BPMN* 1st ed., Heidelberg: dpunkt.verlag GmbH.
- Talib, R. et al., 2010. Agent Assignment for Process Management: Goal Modeling for Continuous Resource Management. In M. zur Muehlen & J. Su, eds. *Business Process Management Workshops*. Sydney, AU: Springer Berlin Heidelberg, pp. 25–36.
- Tanenbaum, A.S., 2007. *Modern Operating Systems* 3rd ed., Prentice Hall.
- Turban, E., Sharda, R. & Delen, D., 2010. *Decision Support and Business Intelligence Systems* 9th ed., Prentice Hall.
- Vaculín, R. et al., 2011. Declarative Business Artifact Centric Modeling of Decision and Knowledge Intensive Business Processes. In *15th IEEE International Enterprise Computing Conference (EDOC 2011)*. Helsinki: IEEE Computer Society, pp. 151–160.
- W3C, Extensible Markup Language (XML) 1.0 (Third Edition). Available at: <http://www.w3.org/TR/2004/REC-xml-20040204>.
- Wahl, M., Howes, T. & Kille, S., 1997. RFC 2251 - Lightweight Directory Access Protocol (v3).
- Warren, D.H.D., 1983. *An Abstract Prolog Instruction Set*. Technical Note 309, SRI International, Menlo Park, CA, USA.
- Weerawarana, S. et al., 2005. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*, Prentice Hall.
- Weske, M., 2012. Business Process Management Methodology. In *Business Process Management - Concepts, Languages, Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 373–388. Available at: <http://www.springerlink.com/index/10.1007/978-3-642-28616-2>.
- Westergaard, M. & Slaats, T., 2013. Mixing Paradigms for More Comprehensible Models. In *BPM 2013*. pp. 283–290.
- Wong, P.Y.H. & Gibbons, J., 2007. A Process Semantics for BPMN. In *Formal Methods and Software Engineering*. Springer, pp. 355–374.
- Workflow Management Coalition, 1999. Workflow Management Coalition: Terminology & Glossary 3.0 (WFMC-TC-1011).
- Workflow Patterns Initiative, 2011. Workflow Patterns Home Page. Available at: <http://www.workflowpatterns.com/>.
- Zeilenga, K., 2006. RFC 4512 - Lightweight Directory Access Protocol (LDAP): Directory Information Models. The Internet Society.

- Zeising, M., Schöning, S. & Jablonski, S., 2012. Improving Collaborative Business Process Execution by Traceability and Expressiveness. In *8th International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012)*. Pittsburgh, PA, US: IEEE Computer Society, pp. 435–442.
- Zeising, M., Schöning, S. & Jablonski, S., 2014. Towards a Common Platform for the Support of Routine and Agile Business Processes. In *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2014)*. Miami, FL, USA: IEEE.

Danksagung

Allen voran gilt mein Dank meinem Doktorvater Prof. Dr.-Ing. Stefan Jablonski, der mir durch das entgegengebrachte Vertrauen die Möglichkeit zur Promotion eröffnet hat. Während meiner Arbeit am Lehrstuhl stand er mir stets mit guten Ratschlägen zur Seite und hat durch zahlreiche Anregungen und Diskussionen wesentlich zur vorliegenden Arbeit beigetragen. Er ermöglichte mir einen intensiven Austausch mit der Forschungsgemeinde und ließ mir den nötigen Freiraum zur persönlichen Entfaltung.

Weiterer Dank gebührt Prof. Dr. Torsten Eymann für die zweite Begutachtung der Arbeit sowie für die wertvollen Anmerkungen und Diskussionen.

Ganz herzlich möchte ich mich auch bei all meinen Kollegen für die zahlreichen fachlichen Gespräche sowie die hilfreiche und konstruktive Kritik bedanken. Namentlich hervorheben möchte ich Lars Ackermann, Florian Gillitzer, Christoph Günther, Matthias Jahn, Bastian Roth und Stefan Schöning zu denen während meiner Zeit am Lehrstuhl ein freundschaftliches Verhältnis entstanden ist, das auch außerhalb der Arbeitszeit besteht. Besonderer Dank gilt hierbei Stefan Schöning, der stets der wertvollste Gesprächspartner und Kritiker war.

Gleichermaßen möchte ich Kerstin Haseloff und Bernd Schlesier danken. Sie ermöglichen stets ein reibungsloses Arbeiten im Lehrstuhllalltag.

Ein großer Dank gilt auch meiner Familie, allen voran meiner Schwester Karoline und meinem Vater Ralf, die auch in Zeiten des Selbstzweifels immer an mich geglaubt haben. Meiner Mutter Kerstin danke ich in dem Glauben, dass sie es ist, die alles in die richtigen Bahnen lenkt. Zu guter Letzt danke ich meiner lieben Maria, die mich ungeachtet ihrer eigenen Belastung auch in den anstrengendsten Zeiten zum Lachen gebracht hat und immer wieder bringt.

*Michael Zeising
Bayreuth, Mai 2015*