

Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diplomarbeit Nr. 3046

Prozessvarianten in unternehmensübergreifenden Servicenetzwerken

Falko Michael Kötter

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Monika Weidmann M.Sc. Dipl.-Inf. Daniel Schleicher
begonnen am:	1. Juni 2010
beendet am:	31. November 2010
CR-Klassifikation:	D.2.4, H.4.1, H.5.3

Zusammenfassung

Das Internet der Dienste ermöglicht eine dynamische Zusammenarbeit von Firmen in unternehmensübergreifenden Geschäftsprozessen. Diese Geschäftsprozesse stellen nicht nur gewachsene Geschäftsbeziehungen, sondern auch Ad-Hoc-Kooperationen dar. Jede Zusammenarbeit verschiedener Partner verlangt eine Konfiguration des Prozesses, doch dies jedes Mal manuell zu modellieren bedeutet einen unvermeidbaren Aufwand. Daraus ergibt sich der Bedarf, unternehmensübergreifende Geschäftsprozesse zeitnah an die Bedürfnisse aller Beteiligten anzupassen.

In dieser Arbeit wird aufbauend auf Praxisanforderungen und bisherigen Ansätzen ein Konzept für Prozessvarianten in unternehmensübergreifenden Servicenetzwerken entwickelt. Dazu wird die Prozessmodellierungssprache *BPMN 2.0* erweitert, sodass zwei neue Diagrammtypen modelliert werden können: *Referenzprozesse* dienen als standardisierter Rahmen für variable Prozesse, während *Prozessfragmente* in diesen Rahmen eingesetzt werden, um Prozessvarianten zu erzeugen. Außerdem werden ein Konzept für variable Attribute von Elementen, für die Modellierung von Abhängigkeiten zwischen einzelnen variablen Teilen des Prozesses sowie für die Generierung einer Variante aus dem Referenzprozess entwickelt. Ein vorhandenes Variabilitätskonzept wird zur Umsetzung und separaten Speicherung der Variabilität genutzt.

Die entwickelten Konzepte werden für den webbasierten Editor *Oryx* umgesetzt. In einem Ausblick werden weiterführende Fragestellungen wie Validierung, Compliance, Simulation und Optimierung betrachtet.

Abstract

In the Internet of Services companies work together using interorganizational business processes. These business processes represent not only long-term business relationships, but also ad-hoc collaborations. Every such cooperation calls for specific adjustments of the process. Implementing these adaptations by hand is too expensive. Therefore, a need for fast adaptability of interorganizational business processes arises.

This paper presents a concept for process variants in interorganizational service networks. To accomplish this, the *Business Process Modeling Notation 2.0* is extended to allow for two new kinds of diagrams: *Process templates* serve as customizable templates for variable processes, in which *process fragments* are inserted to create a specific process variant. Furthermore concepts for variable attributes of modeling elements, dependency modeling between variable process parts as well as for creating a variant from a process template are introduced. An existing variability concept is used to model variability and save it in a separate file.

These concepts are implemented using the web-based editor *Oryx*. Finally, possible future work like validation, compliance, simulation and optimization is outlined.

Inhaltsverzeichnis

1. Einleitung	9
1.1. Aufgabenstellung	10
1.2. Gliederung der Arbeit	10
2. Anforderungen	13
2.1. Variabilität	13
2.2. Sicherheit	14
2.3. Ableitung einer Variante	14
2.4. Modellierungssprache	15
2.5. Ausführbarkeit	16
3. Grundlagen	19
3.1. BPMN	19
3.2. Variabilitätsbeschreibungen	21
3.3. Oryx	25
Stencil Set	27
Stencil Set Extension	28
Plugins	28
4. Vorhandene Ansätze zur Variabilität	31
4.1. Modellierung von Variabilität mittels vorhandener Konstrukte	31
4.2. Provop	32
4.3. PESOA	34
4.4. Prozesskonfigurator	37
4.5. Vergleich der verschiedenen Modellierungsansätze	38
5. Variabilitätskonzept	41
5.1. Variabilitätsmodellierung	41
5.1.1. Variabilitätsumfang	41
5.1.2. Modellierungskonzept	45
Variabler Referenzprozess	47
Prozessfragmente	48
5.1.3. Zusammenfassung	51
5.2. Abhängigkeitsmodellierung	52

5.3. Variantenableitung	56
6. Implementierung	59
6.1. Übersicht	59
6.2. Variabilitätsmodellierung	61
6.2.1. Umsetzung der BPMN-Erweiterung	61
6.2.2. Umsetzung variabler Attribute	61
6.3. Abhängigkeitsmodellierung	63
6.4. Variantenableitung	63
6.4.1. Dependency-Graph	63
6.4.2. Ableitungsmodus	68
6.4.3. Auswertung von Constraints und Enabling Conditions	68
6.4.4. Sidebar	70
6.4.5. Fragment-Import	71
6.4.6. Auflösung der Fragment-Links	72
6.4.7. Automatische Ableitung	72
6.4.8. Protokollierung	73
6.5. Export	73
6.5.1. Variable Attribute	75
6.5.2. Variable Regionen	75
6.5.3. Fragmente	77
6.5.4. Enabling Conditions	79
6.5.5. Abhängigkeiten	79
6.6. Deployment	79
7. Zusammenfassung und Ausblick	81
7.1. Zusammenfassung	81
7.2. Verteilte Modellierung	82
7.3. Variantenausführung	83
7.4. Optimierung	83
7.5. Simulation	85
7.6. Validierung und Compliance	87
A. Anhang	91
A.1. Inhalt und Aufbau des beigelegten Datenträgers	91
A.2. Installation	91
A.3. Anleitung	98
Literaturverzeichnis	113

Abbildungsverzeichnis

3.1. Beispieldiagramm BPMN 2.0	20
3.2. Oryx-Benutzeroberfläche	26
4.1. Option in Provop	33
4.2. PESOA-Beispiel	36
5.1. Variable Region	47
5.2. Fragment-Start und -Ende	49
5.3. Fragment-Link	50
5.4. Auflösung eines Fragment-Links	50
6.1. Architektur	60
6.2. Variables Attribut im Variability Wizard	62
6.3. Variable Attribute	62
6.4. Definition des Dependency-Graph	64
6.5. Dependency-Graph mit und ohne Pivot-Punkt	66
6.6. Prozessfragment	66
6.7. Referenzprozess	67
6.8. Ableitung	67
6.9. Ableitungsmodus	68
6.10. Sidebar-Plugin	71
6.11. Export einer variablen Region	76
7.1. Beispiel BPMN-Q	88

Tabellenverzeichnis

4.1. Vergleich der Modellierungsansätze zur Variabilität	39
5.1. Modellierbarkeit und Umsetzung von Variabilität	52

Verzeichnis der Listings

3.1. Aufbau eines Variability Descriptors (nach [Mieo8])	21
3.2. Definition von Locators für einen Variability Point (nach [Mieo8])	22
3.3. Definition von Alternativen für einen Variability Point (nach [Mieo8])	22
3.4. Explizite Alternative (nach [Mieo8])	23
3.5. Explizite Alternative (nach [Mieo8])	23
3.6. Leere Alternative (nach [Mieo8])	23
3.7. Ausdrucksalternative (nach [Mieo8])	24
3.8. Freie Alternative (nach [Mieo8])	24
3.9. Definition einer Dependency (nach [Mieo8])	24
5.1. Attributsprache	51
5.2. Um Abhängigkeiten erweiterte Attributsprache, hinzugefügte Teile sind hervorgehoben	54
6.1. Algorithmus Fragmentwahl in der exportierten Variabilitätsbeschreibung . . .	76

1. Einleitung

Im Zuge von Software-As-A-Service wird Software zunehmend als Dienstleistung angeboten. Im Rahmen der serviceorientierten Architektur (SOA) werden neue Anwendungen aus solchen Services zusammengesetzt. Ein Beispiel dafür sind Geschäftsprozesse, die einen Ablauf in einem Unternehmen beschreiben, z.B. die Bestellung in einem Online-Shop oder die Entgegennahme einer Reklamation.

Zunehmend verlaufen diese Geschäftsprozesse zwischen mehreren Teilnehmern, die einzelne Unternehmensteile, aber auch einzelne Firmen darstellen können. Beispiele dafür finden sich in der Versicherungsbranche, wo bei der Abwicklung eines Schadensfalls neben Versicherung und Kunde auch externe Gutachter und Handwerker beteiligt sind. Diese einzelnen Teilnehmer müssen nicht jedes Mal die selben sein, sondern können auch von Prozessinstanz zu Prozessinstanz variieren. Bei einem Schadensfall einer Kfz-Versicherung bietet es sich beispielsweise an, einen Gutachter einzusetzen, der sich vor Ort befindet.

Bei dieser erhöhten Dynamik von Geschäftspartnern spricht man vom Internet der Dienste[Scha]. Die Vision ist, dass Dienstleistungen ebenso wie oben beschrieben Software über das Internet automatisiert angeboten und gekauft werden können. Während es jedoch bei Software einfach möglich ist, Schnittstellen zu definieren, stellt es sich bei Firmen anders dar. Verschiedene Versicherungsunternehmen haben zum Beispiel verschiedene Regeln, wann sie einen Gutachter hinzuziehen und Gutachter haben verschiedene Leistungen im Angebot.

Jede einzelne Zusammenarbeit verschiedener Partner stellt einen neuen Prozess dar, doch diesen jedes Mal von Hand zu modellieren bedeutet einen unvermeidbaren Aufwand. Solche unternehmensübergreifenden Geschäftsprozesse erfordern also eine gewisse Anpassbarkeit, um die reibungslose Zusammenarbeit mit verschiedenen Dienstleistern zu ermöglichen. Diese Anpassbarkeit kann dadurch erreicht werden, dass Teile des Prozesses variabel sind, also an die Gegebenheiten der spezifischen Geschäftsbeziehung angepasst werden können. Andere Teile sollen standardisiert sein. Man könnte also von einem Mass-Customizing für Geschäftsprozesse sprechen.

1.1. Aufgabenstellung

Ziel dieser Arbeit ist es, ein Modell zur Prozesskonfiguration für unternehmensübergreifende Servicenetzwerke zu entwickeln und zu implementieren. Als Grundlage dafür dient ein Praxisprojekt, das sich mit unternehmensübergreifenden Servicenetzwerken in der Versicherungsbranche beschäftigt.

Grundlage für die Erstellung des Modells sind die aus dem Praxisprojekt stammenden Anforderungen und Rahmenbedingungen sowie der Stand der Forschung. Um diese zu erarbeiten, sollen die Anforderungen zunächst zusammengefasst werden und unter ihrer Berücksichtigung eine State-Of-The-Art-Analyse bisheriger Entwicklungen durchgeführt werden.

Basierend auf den gewonnenen Erkenntnissen soll ein Konzept zur Variabilitätsmodellierung entwickelt werden, das auf der *Business Process Modeling Notation* aufbaut und in der Lage ist, die erarbeiteten Anforderungen zu erfüllen.

Dieses Modell soll dann prototypisch in einem vorhandenen BPMN-Editor umgesetzt werden. Zusätzlich soll eine Integration mit einem existierenden Variabilitätswerkzeug, den sog. *Variabilitätsbeschreibungen* ermöglicht werden.

Im Ausblick sollen Möglichkeiten zur Ausführbarkeit der Modelle, Validierung, Simulation und Optimierung aufgezeigt werden.

1.2. Gliederung der Arbeit

Kapitel 1 gibt eine Einleitung in das Thema der Arbeit und beschreibt die Aufgabenstellung.

In Kapitel 2 werden die Anforderungen an eine Lösung zur Prozesskonfiguration anhand eines Praxisprojekts erarbeitet.

Kapitel 3 enthält Grundlagen, die für das weitere Verständnis der Arbeit notwendig sind. Unter anderem werden Variabilitätsbeschreibungen (3.2), die Business Process Modeling Language (3.1) und der Diagrammeditor Oryx (3.3) behandelt.

Als Grundlage für ein eigenes Variabilitätskonzept werden in Kapitel 4 vorhandene Ansätze zur Variabilitätsmodellierung vorgestellt und miteinander verglichen.

Darauf aufbauend wird in Kapitel 5 ein Variabilitätskonzept erarbeitet, das die Variabilitätsmodellierung (5.1), die Abhängigkeitsmodellierung (5.2) und die Variantenableitung (5.3) umfasst.

Dieses Variabilitätskonzept dient als Basis für die Implementierung, die in Kapitel 6 beschrieben wird. Die Implementierung erweitert den Oryx-Editor, sodass Variabilitätsmodellierung und Variantenableitung ermöglicht werden. Außerdem wird ein Export des variantenreichen Modells als Variabilitätsbeschreibung ermöglicht.

Abschließend fasst Kapitel 7 die Ergebnisse der Arbeit zusammen und bietet einen Ausblick auf weiterführende Ansätze und zukünftige Entwicklungen.

2. Anforderungen

Anforderungen für die Umsetzung eines unternehmensübergreifenden Prozesses werden wie in der Aufgabenstellung beschrieben anhand eines Kooperationsprojektes des Fraunhofer IAO, des IAT der Universität Stuttgart und der Metris GmbH ermittelt. Das Projekt *openX-change* dient der Schaffung eines Marktplatzes im Rahmen der Schadensregulierung von Versicherungen[WWWr]. Am Schadensregulierungsprozess sind verschiedene Unternehmen beteiligt. Neben den Versicherungsunternehmen sind das Handwerksunternehmen und Sachverständige, die im Rahmen der Regulierung von Versicherungen beauftragt werden.

Neben der Vermittlung der Akteure soll *openXchange* auch die elektronische Durchführung des Regulierungsprozesses ermöglichen[WWWr]. Dabei soll die Prozesschoreographie zwischen den einzelnen Akteuren von der Plattform übernommen werden. Dieser Prozess muss natürlich anhand von unternehmensinternen Richtlinien, geschlossenen Verträgen, Service-Level-Agreements und Ähnlichem anpassbar sein.

Da die Plattform als Anbieter für die Prozessdurchführung auftritt, muss sie es den Nutzern ermöglichen, ihren Teil des Prozesses dort zu modellieren oder zumindest zu hinterlegen. Die Interaktion zwischen den einzelnen Teilnehmern wird vom Plattformbetreiber modelliert. Dieses übergreifende Modell dient als eine Art Interface zwischen den Kunden und kann von ihnen nicht verändert werden.

2.1. Variabilität

Um dennoch eine gewisse Flexibilität zu ermöglichen, soll der Standardprozess generisch modelliert werden, d.h. er soll Variabilität enthalten.

Beispiele für die Anwendung von Variabilität finden sich im Gebiet des Software Product Family Engineering. Dieser Bereich der Informatik befasst sich mit der Erstellung und Pflege von mehreren Varianten ein und desselben Softwareprodukts, die eine gemeinsame Codebasis haben[PBL05]. Ein Beispiel hierfür ist ein Betriebssystem, das in Varianten für Heimanwender und Geschäftsnutzer angeboten wird.

Zur Umsetzung von Variabilität können sogenannte *Variability Points* oder auch *Variation Points*, zu deutsch *Variabilitätspunkte* verwendet werden. Ein Variation Point ist ein konkreter Punkt innerhalb eines Produkts, der Variabilität unterliegt.

In [PBL05] wird ein Variation Point wie folgt definiert:

Definition 1. *A variation point is a representation of a variability subject within domain artefacts enriched by contextual information.*

Ein generischer Prozess, der solche Variabilität enthält, wird *Referenzprozess* genannt.

Um die Modellierung von Variabilität zu ermöglichen, muss eine Modellierungssprache um entsprechende Konstrukte erweitert werden. Dafür gibt es bereits einige Ansätze, die in Kapitel 4 näher vorgestellt werden.

Zur Speicherung der Variabilität und für die Ableitung existiert bereits ein Format, die sogenannten *Variabilitätsbeschreibungen*, die in dieser Arbeit Verwendung finden sollen. Diese werden in 3.2 näher erläutert.

Die Erstellung einer konkreten Variante aus dem variablen Dokument wird als *Ableitung* bezeichnet. Im Laufe dieser Ableitung werden die *Variability Points* mit konkreten Werten belegt (sogenanntes Binding)[Mieo8]. Das Ergebnis solch einer Ableitung heißt *Variante*.

2.2. Sicherheit

Da bei *openXchange* mehrere Parteien an der Modellierung und Durchführung eines Prozesses beteiligt sind, ist es notwendig, deren Zuständigkeiten voneinander zu trennen. Bei der Modellierung wird zwischen den Rollen des Referenzprozessmodellierers und des Plattformteilnehmers unterschieden werden.

Der Referenzprozessmodellierer erstellt den eigentlichen Referenzprozess, der als Rahmen für die Interaktion dient. Plattformteilnehmer können ihn wie oben beschrieben während der Variantenableitung an die eigenen Bedürfnisse anpassen.

Diese Anpassbarkeit reicht allerdings nicht aus. Möchte beispielsweise ein Versicherungsunternehmen innerhalb seines Teils des Prozesses eigene Geschäftslogik einführen, so muss diese ebenfalls modelliert sein, damit sie im Referenzprozess ausgewählt werden kann. Dazu muss auch der Prozessteilnehmer die Möglichkeit haben, eigene Teile des Referenzprozesses zu modellieren. Die Teile des Referenzprozesses, die in der Verantwortung der Plattform oder seiner Geschäftspartner liegen, soll er hingegen nicht verändern dürfen.

2.3. Ableitung einer Variante

Werden über die Plattform mehrere Akteure für einen Prozess vermittelt, so muss aus dem Referenzprozess eine konkrete Variante erstellt werden. Dies soll für die Nutzer der Plattform so einfach wie möglich geschehen. Zum Teil sind bei der Plattform bereits Daten hinterlegt,

mittels denen die Entscheidung für einige Variabilitätspunkte getroffen werden müssen. Zum anderen sollen die relevanten Entscheidungen von den Beteiligten getroffen werden, die dafür zuständig sind.

Dabei muss sichergestellt werden, dass in jedem Fall eine lauffähige Variante entsteht, es also keine Widersprüche zwischen den einzelnen Entscheidungen gibt. Ein Beispiel für solch einen Widerspruch wäre, dass ein Versicherungsunternehmen auf einen Kostenvoranschlag wartet, den das Handwerksunternehmen laut Prozess nicht verschickt.

Ein Mittel, um dem Modellierer des Referenzprozesses den Ausschluss solcher Widersprüche zu ermöglichen, soll die Modellierung von Abhängigkeiten zwischen Variabilitätspunkten sein. Eine solche Abhängigkeit wäre beispielsweise, dass das Versicherungsunternehmen die Variante Kostenvoranschlag nur wählen darf, wenn das Handwerksunternehmen dies ermöglicht.

Bei der Ableitung des Prozesses können diese Abhängigkeiten dann verwendet werden, um gewisse Entscheidungen automatisch auf Basis anderer Entscheidungen zu treffen. Wird beispielsweise ein Kostenvoranschlag vom Versicherungsunternehmen gewählt, wird dessen Erstellung automatisch ebenfalls im Prozess des Handwerksunternehmens berücksichtigt.

Eine Ableitung mit inkompatiblen Präferenzen kann allerdings nicht ausgeführt werden. Verlangt das Versicherungsunternehmen beispielsweise einen Kostenvoranschlag und führt das Handwerksunternehmen grundsätzlich keinen durch, so ist die Erstellung eines lauffähigen Prozesses nicht möglich. Die Zusammenarbeit solch inkompatibler Partner muss von der Plattform bereits in der Vermittlungsphase ausgeschlossen werden und ist nicht Bestandteil dieser Arbeit.

Für die Ableitung einer Variante existiert bereits ein Werkzeug[Aueo8], das Variabilitätsbeschreibungen verwendet. Deswegen soll ermöglicht werden, erstellte Diagramme als Variabilitätsbeschreibung zu exportieren.

2.4. Modellierungssprache

Da die Modellierung sich an Geschäftskunden richtet, ist die Vorgabe für die zu erweiternde Modellierungssprache die Business Process Modeling Notation, die in 3.1 näher beschrieben wird. Die aktuelle Version dieses Standards ist 1.2[WWWs]. Eine neue Version 2.0 liegt als sog. Beta in einer vorläufigen Fassung vor, wird allerdings in dieser Form schon von Werkzeuganbietern unterstützt.

Für diese Arbeit relevante Neuerungen in BPMN 2.0 gibt es in drei Bereichen: Prozessschonographie, Serialisierung und Ausführbarkeit. Im Folgenden werden BPMN 1.2 und BPMN 2.0 in Bezug auf diese drei Merkmale verglichen.

Prozesschoreographie BPMN 2.0 enthält zwei neue Diagrammtypen für Prozesschoreographien, nämlich das *Conversation Diagramm*, mit dem festgelegt wird, welche Konversationen zwischen welchen Partnern ablaufen, und das *Choreography Diagramm*, mit dem der genaue Ablauf einer einzelnen Konversation beschrieben wird.[Schb]

Serialisierung BPMN 1.2 beinhaltet kein eigenes Serialisierungsmodell. Für die Serialisierung kann allerdings die XML Process Definition Language (kurz XPDL) verwendet werden[Allo9], die sich als De-Facto-Standard durchgesetzt hat. BPMN 2.0 dagegen bringt ein eigenes Serialisierungsformat mit, das sogenannte BPMN Diagram Interchange XML.

Ausführbarkeit BPMN 1.2 ist nicht ausführbar. Die Spezifikation skizziert zwar eine Übersetzung nach BPEL[OMG09a], diese führt aber aufgrund der Unterschiede zwischen beiden Sprachen zu komplexen und unverständlichen Diagrammen[OADAO6]. Im Gegensatz dazu behandelt die Spezifikation von BPMN 2.0 die Ausführbarkeit von Prozessen[OMG09b].

Wegen dieser Neuerungen wird für diese Arbeit BPMN 2.0 verwendet.

2.5. Ausführbarkeit

Ein abgeleiteter Prozess muss ausführbar sein. Dies kann durch unmittelbare Ausführung des Prozesses mit einer Business Process Engine oder durch Umwandlung in eine andere Sprache (z.B. WS-BPEL) geschehen.

Die Frage, die sich stellt, ist, wo welche Teile des unternehmensübergreifenden Geschäftsprozesses ausgeführt werden. Denkbar sind folgende drei Alternativen:

- Der Referenzprozess wird in Abstimmung aller Beteiligten abgeleitet und die resultierende Variante auf der Plattform ausgeführt. Beteiligte haben die Möglichkeit, eigene Teile des Workflows auf der Plattform zu modellieren.
- Der Referenzprozess enthält nur die Teile des Prozesses, die für die Zusammenarbeit aller Beteiligten notwendig sind. Die internen Abläufe der einzelnen Prozessteilnehmer sind nicht auf der Plattform hinterlegt. Bei der Ableitung ergibt sich eine Variante, die nur ein Interface darstellt, das den Ablauf zwischen den einzelnen Teilnehmern koordiniert. Diese haben eigene Process Engines, die mit der Plattform kommunizieren.
- Der Referenzprozess wird zwar auf der Plattform zu einer Variante abgeleitet, diese wird aber von einem der Beteiligten heruntergeladen und in einer eigenen Process Engine ausgeführt.

Die letzten beiden Alternativen haben den Nachteil, dass die Plattform die Kontrolle über den ausgeführten Prozess verliert, wodurch unter Anderem die Ausführungsqualität, Sicherheit und Integrität des Prozesses nicht mehr garantiert werden können.

Deswegen und da *openXchange* als Vermittler und zentrale Dienstplattform für alle Teilnehmer konzipiert ist, wird die erste Alternative gewählt.

Der BPMN 2.0 Standard ist zum Zeitpunkt dieser Arbeit noch nicht fertiggestellt. Process Engines, die die Ausführung von BPMN 2.0 ermöglichen, befinden sich noch in der Entwicklung. Aus diesen Gründen ist der Fokus in der Aufgabenstellung auf die Modellierung gelegt. Die Ausführbarkeit der Varianten wird in dieser Arbeit nur im Ausblick behandelt.

Um die spätere Integration mit einer Process Engine zu erleichtern, soll diese allerdings in der zu entwickelnden Lösung berücksichtigt werden. Denkbar wäre dies beispielsweise durch eine Deployment-Funktion, die die fertige Variante in einem Exportformat bereitstellt, sodass diese nur noch an die Process Engine übergeben werden muss.

3. Grundlagen

In diesem Kapitel sollen dem Leser einige Grundlagen vermittelt werden, die zum Verständnis der Arbeit notwendig sind. Es werden drei Technologien behandelt, die im weiteren Verlauf der Arbeit verwendet werden. Dies sind die verwendete Modellierungssprache *BPMN*, *Variabilitätsbeschreibungen* zur getrennten Speicherung von Variabilität und die web-basierte Modellierungsumgebung *Oryx*, die als Grundlage für die Implementierung der Lösung dient.

3.1. BPMN

Die *Business Process Modeling Notation*[WWWs] (kurz BPMN) ist eine Sprache zur Geschäftsprozessmodellierung, die von einem Konsortium namens BPML.org unter Führung von Intalio entwickelt wurde[Siloga]. Zweck der Sprache ist es, Personen ohne technischen Hintergrund wie z.B. Geschäftsanwendern zu ermöglichen, ihre eigenen Prozesse zu modellieren.

Der BPMN-Standard wird inzwischen von der Object Management Group (OMG) beaufsichtigt und steht in der aktuellen Version 1.2 zur Verfügung. Version 2.0 befindet sich gerade in der Finalisierungsphase[Siloga] und wird bereits von einigen Werkzeugen unterstützt[WWWk].

Im Folgenden sollen kurz Grundzüge der BPMN, konkret von Kollaborationsdiagrammen, vorgestellt werden. Detailliertere Informationen finden sich z.B. in [Siloga] und [WWWb] als Überblick.

Ein Kollaborationsdiagramm besteht grundlegend aus drei Arten von Elementen, auch Flow Objects genannt[Siloga]:

Activity Eine Activity (dt. Aktivität) wird als abgerundetes Rechteck dargestellt. Sie beschreibt eine durchzuführende Tätigkeit. Beispiele für eine Aktivität sind ein Task oder auch ein Subprozess, der seinerseits einen kompletten Prozess enthält.

Gateway Ein Gateway wird als Raute dargestellt und ist eine Verzweigung im Sequenzfluss. Je nach Art des Gateways können ein, mehrere oder alle ausgehenden Pfade gewählt werden. Mit einem Gateway kann somit Nebenläufigkeit modelliert werden. Treffen sich die alternativen Pfade wieder, wird ebenfalls ein Gateway verwendet.

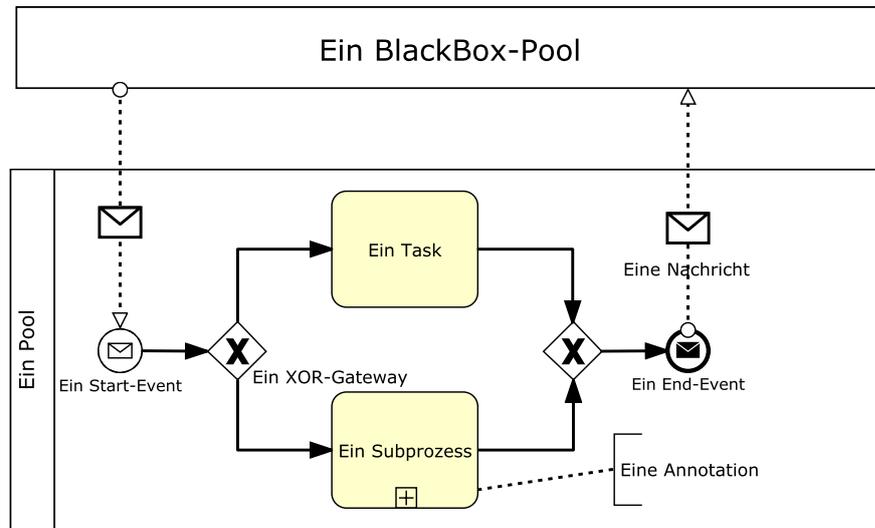


Abbildung 3.1.: Beispieldiagramm. Der Prozess beginnt mit Empfang einer Nachricht, führt dann entweder den Task oder den Subprozess durch und endet mit dem Versand einer Nachricht an den Prozessinitiator.

Event Ein Event repräsentiert etwas, das im Prozess geschieht und wird als Kreis dargestellt. Events sind beispielsweise Nachrichten, Signale, Time-Outs und Fehler. Ein Event kann z.B. erzeugt oder aber auch erwartet werden. In Abbildung 3.1 wird der Prozessbeginn beispielsweise durch eine eingehende Nachricht begonnen. Am Ende wird eine Nachricht versendet.

Um die Kollaboration zu modellieren, werden die Prozesse der einzelnen Akteure in sogenannten *Pools* dargestellt. Ein Pool repräsentiert dabei eine Organisation. Kommunikation zwischen Pools findet über Messages statt, denn der Sequenzfluss darf die Grenzen eines Pools nicht überschreiten. Ist der Prozess eines Akteurs nicht bekannt, so kann er auch als Black Box modelliert werden. Der Pool kann in mehrere *Lanes* unterteilt werden, die die einzelnen Teile einer Organisation darstellen. Kommunikation zwischen einzelnen Lanes verläuft per Sequenzfluss, Nachrichten dürfen nicht ausgetauscht werden.

Zur Annotierung der Elemente stehen *Annotations* zur Verfügung, die ähnlich wie Quelltextkommentare in Programmiersprachen keine Auswirkungen auf die Ausführung haben.

3.2. Variabilitätsbeschreibungen

Variabilitätsbeschreibungen[Mie10] (engl. Variability Descriptors) dienen der Beschreibung von Variability Points in beliebigen Dokumenten. Die Besonderheit des Ansatzes ist, dass die Variabilität vom ursprünglichen Dokument getrennt wird und somit eine Verwendung ermöglicht wird, ohne die Sprache des Dokuments (z.B. WS-BPEL) um Konstrukte für Variabilität zu erweitern[MLo8].

Eine Variabilitätsbeschreibung enthält eine Menge von Variability Points und erfüllt drei Hauptzwecke[Mieo8]:

Kenntlichmachung Ein Variability Point macht die variable Stelle eindeutig kenntlich.

Festlegung des Wertebereichs Der Wertebereich für mögliche Ableitungen eines Variability Point wird spezifiziert.

Beschreibung von Abhängigkeiten Abhängigkeiten zwischen Variability Points dienen zum einen der Festlegung einer Reihenfolge bei der Variantenableitung, zum anderen der Beschränkung des Wertebereichs durch vorherige Entscheidungen. Wenn beispielsweise eine zusätzliche Prüfung abgelehnt wird, muss für diese Prüfung kein Zuständiger festgelegt werden.

Im Folgenden werden die Möglichkeiten des Variability Descriptors anhand seiner Grundstruktur erläutert:

```
<?xml version="1.0" encoding="UTF-8"?>
<variabilityDescriptor
xmlns="http://www.iaas.uni-stuttgart.de/schemas/VD"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.iaas.uni-stuttgart.de/schemas/VD VD.xsd"
expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"
targetNamespace="http://www.iaas.uni-stuttgart.de/schemas/taasVD">
  <variabilityPoints>
    <variabilityPoint name="VP1">
      <locators>
        ...
      </locators>
      <alternatives>
        ...
      </alternatives>
    </variabilityPoint>
  </variabilityPoints>
  <dependencies>
    ...
  </dependencies>
</variabilityDescriptor>
```

Listing 3.1: Aufbau eines Variability Descriptors (nach [Mieo8])

3. Grundlagen

Zur Erfüllung der drei oben aufgeführten Anforderungen enthält der Variability Descriptor dreierlei Elemente, *Locators*, *Alternatives* und *Dependencies*, eines für jede der Anforderungen.

Die ersten beiden beschreiben einen einzelnen `VariabilityPoint`. Ein `Variability Point` wird mittels des Attributs `name` eindeutig benannt.

Locators

```
<locators>
  <locator> document("model.bpmn")/definitions/process//Task[@name="VP1"/
    </locator>
</locators>
```

Listing 3.2: Definition von Locators für einen Variability Point (nach [Mie08])

Das erste Element ist `locators`, mittels dessen alle Stellen im Dokument angegeben werden, an denen der `Variability Point` das Originaldokument verändert. Jeder einzelne dieser Punkte ist durch ein `locator`-Element eindeutig festgelegt. Dazu enthält der `Locator` eine Adresse, beispielsweise in der *XML Path Language (XPath)*[WWV]. Möchte man eine andere Sprache zur Adressierung verwenden, so kann diese mittels des optionalen Attributs `expressionLanguage` im `locator`-Element gewählt werden.

Alternatives

```
<alternatives>
  <alternative name="Alternative1" default="true"> ... </alternative>
  <alternative name="Alternative2"> ... </alternative>
  <alternative name="Alternative3"> ... </alternative>
</alternatives>
```

Listing 3.3: Definition von Alternativen für einen Variability Point (nach [Mie08])

Das zweite Element des `Variability Points` ist `alternatives`, in dem sich die verschiedenen Alternativen für die Ableitung des `Variability Point` befinden. Ein einzelnes `alternative`-Element setzt sich wie folgt zusammen:

Mittels des Attributs `name` erhält die `Alternative` einen innerhalb des `Variability Points` eindeutigen Namen. Zudem kann eine einzige `Alternative` pro `Variability Point` mittels des Attributs `default` als voreingestellte `Alternative` gewählt werden.

Um die `Alternative` selbst zu beschreiben, stehen fünf verschiedene Arten von Alternativen zur Verfügung.[Aue08]

explicitAlternative Die explizite `Alternative` ermöglicht es, als `Alternative` einen festen Text vorzugeben.

expressionAlternative Ähnlich wie beim `locator`-Element kann in der `Ausdrucksalternative` eine Adresse angegeben werden, die auf eine `Alternative` verweist.

freeAlternative Mittels der freien Alternative kann ein Variability Point mit einem beliebigen Freitext abgeleitet werden.

locatorAlternative Mittels der Zeiger-Alternative kann der bestehende Wert des Variability Point als Alternative spezifiziert werden.

emptyAlternative Die leere Alternative ermöglicht es, den Variability Point mit einem leeren String zu belegen.

Im Folgenden werden die Alternativen näher erläutert.

Explizite Alternative

```
<alternative name="MyExplicitAlternative">
  <explicitAlternative>
    <task completionQuantity="1" startQuantity="1"
      isForCompensation="false" name="Fehlerbehandlung" id="task_17"/>
  </explicitAlternative>
</alternative>
```

Listing 3.4: Explizite Alternative (nach [Mieo8])

Die `explicitAlternative` enthält ein Kindelement, das vom Typ `xsd:any` ist, sodass man beliebige Inhalte angeben kann. Dieser Inhalt wird dann als Alternative zur Auswahl gestellt. In Listing 3.4 ist das zum Beispiel ein BPMN-Task.

Zeiger-Alternative

```
<alternative name="MyLocatorAlternative">
  <locatorAlternative/>
</alternative>
```

Listing 3.5: Explizite Alternative (nach [Mieo8])

Die Zeiger-Alternative ermöglicht es, den Wert, der bereits am Variability Point steht, als Alternative anzubieten. Wird diese Alternative gewählt, werden bei der Ableitung des Variability Point keine Änderungen am Originaldokument vorgenommen. Zeigt der Variability Point auf mehrere Stellen im Dokument, so wird bei jedem von ihnen der bestehende Wert unverändert gelassen.[Mieo8]

Leere Alternative

```
<alternative name="MyEmptyAlternative">
  <emptyAlternative/>
</alternative>
```

Listing 3.6: Leere Alternative (nach [Mieo8])

Mit der leeren Alternative kann die Ableitung des Variability Point mit einem leeren String zur Auswahl gestellt werden. Dies ist nicht zu verwechseln mit der Zeiger-Alternative, die den Variability Point nur unverändert lässt. Die leere Alternative hingegen löscht einen eventuell vorhandenen Wert aus dem Dokument.

Ein Variability Point darf maximal eine leere Alternative enthalten.[Aueo8]

Ausdrucksalternative

```
<alternative name="MyExpressionAlternative">
  <expressionAlternative>
    document("model.bpmn")/definitions/process//Task/
  </expressionAlternative>
</alternative>
```

Listing 3.7: Ausdrucksalternative (nach [Mieo8])

Die Ausdrucksalternative ermöglicht es, eine Alternative aus dem Originaldokument zu verwenden. Dies wird wie bei einem Locator mittels einer Adresse ermöglicht, an der sich die Alternative befindet. Verweist diese Adresse auf mehrere Elemente, werden alle als Alternative wählbar.

In Listing 3.7 beispielsweise werden alle Tasks als Alternativen zur Auswahl gestellt.

Freie Alternative

```
<alternative name="MyFreeAlternative">
  <freeAlternative>$value &gt;=1 and $value &lt;= 1000</freeAlternative>
</alternative>
```

Listing 3.8: Freie Alternative (nach [Mieo8])

Die freie Alternative ermöglicht es, beim Ableiten der Variante einen beliebigen Text anzugeben. Dieser Freitext kann allerdings auch durch *Constraints* eingeschränkt werden, um beispielsweise die Angabe von einer Zahl aus einem Intervall zu ermöglichen. Dazu kann die Variable `$value` verwendet werden, mittels der man Constraints für den eingegebenen Wert definieren kann.[Mieo8]

In Listing 3.8 werden die Constraints so definiert, dass eine Zahl zwischen 1 und 1000 eingegeben werden muss.

Dependencies

```
<dependency>
  <sources>
    <variabilityPointRef>tvd:VarPoint1</variabilityPointRef>
    <variabilityPointRef>tvd:VarPoint2</variabilityPointRef>
    ...
  </sources>
  <target>
    <variabilityPointRef>tvd:VarPoint3</variabilityPointRef>
```

```

</target>
<enablingConditions>
  <enablingCondition>
    <condition>selectedAlternative("tvd:VP1", "tvd:Alternativ1")</condition>
    <enabledAlternatives>
      <alternativeRef>tvd:alt1</alternativeRef>
      <alternativeRef>tvd:alt2</alternativeRef>
    </enabledAlternatives>
  </enablingCondition>
  ...
</enablingConditions>
</dependency>

```

Listing 3.9: Definition einer Dependency (nach [Mieo8])

Mittels Dependencies können Abhängigkeiten zwischen einzelnen Variability Points definiert werden. Diese Abhängigkeiten werden mittels einer Quelle-Ziel-Beziehung spezifiziert. Für einen `target`-Variability Point können mehrere `sources` angegeben werden, von denen der `target`-Point abhängig ist. Eine solche Beziehung bedeutet, dass der als `target` spezifizierte Punkt von allen unter `sources` definierten Punkten abhängig ist. Diese müssen also vor ihm abgeleitet werden.[MLo8]

In Listing 3.9 wird beispielsweise beschrieben, dass `VarPoint3` abhängig von `VarPoint1` und `VarPoint2` ist.

Diese Abhängigkeiten erlauben es nur, die Ableitungsreihenfolge zu bestimmen. Soll außerdem in Abhängigkeit von vorherigen Entscheidungen der Wertebereich des Variability Point eingeschränkt werden, können `enablingConditions` eingesetzt werden, mittels denen Vorbedingungen für einzelne Alternativen definiert werden können.[Mieo8] Eine `enablingCondition` besteht aus zwei Teilen, der `condition` und den `enabledAlternatives`. Mittels der `condition` wird eine logische Bedingung angegeben, beispielsweise, dass bei einem vorherigen Variability Point eine bestimmte Alternative gewählt wurde. Ist diese Bedingung erfüllt, werden die in `enabledAlternatives` angegebenen Alternativen freigegeben. Eine leere `enablingCondition` bedeutet, dass die Alternativen immer freigegeben sind. Sie darf nur ein einziges Mal pro `dependency` verwendet werden.[Aueo8]

3.3. Oryx

Oryx[WWWa] ist ein web-basierter BPMN-Editor, der vom Hasso-Plattner-Institut in Potsdam entwickelt wurde. Oryx steht unter der MIT License und ist damit Open Source[WWWf]. Die Firma Signavio bietet allerdings auch eine kommerzielle Variante namens Signavio Process Editor an, die auch als Service angeboten wird[WWWn].

3. Grundlagen

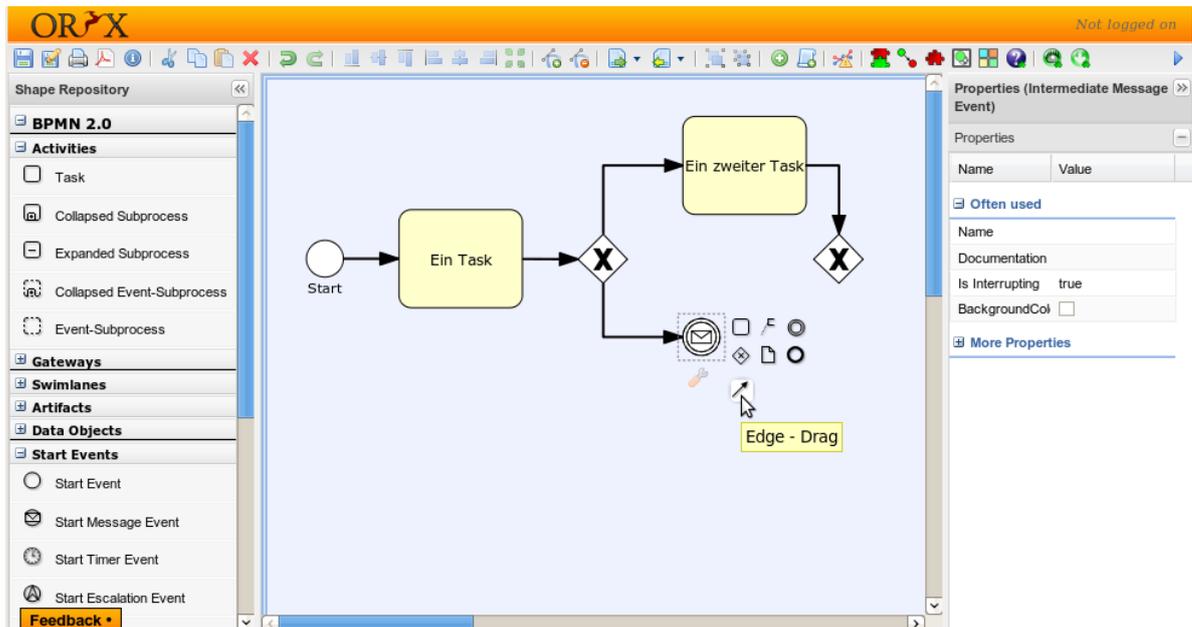


Abbildung 3.2.: Die Oryx-Benutzeroberfläche.

Abbildung 3.2 zeigt die Benutzeroberfläche des Oryx-Editors: Links die Auswahl der Modellierungselemente, in der Mitte die *Canvas* zur Modellierung, rechts die Attribute des gewählten Elements, darüber die Werkzeugleiste.

Oryx bietet mehrere Möglichkeiten für Erweiterungen:

Stencil Sets Ein StencilSet ist ein Set von Darstellungselementen mit zugehörigen Regeln, die unter Anderem erlauben, festzulegen, welche Elemente mit welchen verbunden sein können. Mit einem StencilSet lässt sich eine neue Modellierungssprache definieren und im Oryx-Editor verwenden.[Pet07]

Stencil Set Extensions Möchte man keine neue Modellierungssprache erstellen, sondern nur eine vorhandene erweitern, kann man eine StencilSetExtension verwenden. Wie bei der Vererbung in der objektorientierten Programmierung leitet sich die Extension von einem bestehenden StencilSet ab, das mittels eines Keyword *extends* definiert wird.[Pet07]

Client-Plugins Client-Plugins werden mit dem Editor im Browser des Clients geladen und können zusätzliche Funktionalität zur Verfügung stellen. Über eine Fassade kann ein Plugin auf die Funktionen und Daten des Editors zugreifen. Mittels Events können Plugins miteinander kommunizieren und auf Nutzerverhalten reagieren.[WB10]

Server-Plugins Oryx erlaubt die Verwendung von Servlets als Server-Plugins.

Da diese Möglichkeiten die Grundlage für die zu entwickelnde Lösung bilden, werden sie im Folgenden noch einmal näher vorgestellt.

Stencil Set

Ein StencilSet besteht aus sogenannten Stencils, die einzelne Formen der umzusetzenden Modellierungssprache darstellen. Grundsätzlich werden zwei Arten von Stencil unterschieden:

Node Unter Node werden alle Elemente zusammengefasst, die nicht zur Verbindung anderer Elemente dienen. In BPMN sind dies beispielsweise Tasks und Subprozesse.

Edge Unter Edge versteht man die Elemente, die der Verbindung von Diagrammelementen dienen. In der Regel sind dies Pfeile oder Ähnliches wie beispielsweise in BPMN Sequenz- und Nachrichtenfluss.

Das StencilSet setzt sich aus einer Spezifikation dieser Stencils und zugehörigen Grafiken zusammen.

Die Beschreibung des StencilSet erfolgt in JSON (*Java Script Object Notation*), einem auf JavaScript basierendem Format zum Datenaustausch[WWWc]. In dieser Beschreibung besteht ein einzelner Stencil aus Attributen wie z.B. sein Name und seine Beschreibung sowie frei definierbaren *Properties*, mittels denen definiert wird, welche Attribute der Stencil im Editor hat. Beispiele für solche Attribute in BPMN sind der Name eines Elements oder der Typ eines Tasks. Diese Properties haben verschiedene Datentypen (z.B. String, Boolean oder Enumeration) und können vom Nutzer mit angemessenen Bedienelementen (z.B. Textfeld, Checkbox, Drop-Down-Box) bearbeitet werden. Außerdem kann mittels einer Referenz definiert werden, wie diese Properties grafisch dargestellt werden. So lassen sich beispielsweise abhängig von gewählten Optionen einzelne Teile der grafischen Repräsentation ein- und ausschalten.[Pet07]

Für die Darstellung der Elemente wird das Vektorgrafik-Format SVG (*Scalable Vector Graphics*) verwendet, das für Oryx um einige Elemente erweitert wurde. So können beispielsweise einzelne Teile der Grafik ein- und ausgeblendet werden sowie Text modifiziert werden. Außerdem können an Teilen der Grafik sogenannte *Magnets* definiert werden, an denen Kanten automatisch angedockt werden. Mittels der Erweiterung kann spezifiziert werden, wie sich ein Element verhält, wenn seine Größe verändert wird.[Pet07]

Für die Beziehungen zwischen den einzelnen Elementen können verschiedene Arten von Regeln definiert werden:

Connection Rules Mittels dieser Regeln kann beschrieben werden, welche Elemente mit welchen verbunden werden dürfen.

Cardinality Rules Diese Regeln dienen dazu, Beschränkungen der Anzahl von Elementen und Verbindungen anzugeben. So kann z.B. festgelegt werden, dass ein Element eine oder keine ausgehende Verbindung haben darf oder nur exakt einmal im Diagramm vorkommen darf.

Morphing Rules Diese Regeln erlauben die Definition von Gruppen, deren Elemente beim Editieren ineinander umgewandelt werden dürfen[Kun09]. Ein Beispiel dafür sind in BPMN die verschiedenen Event-Arten (Nachricht, Fehler, etc.).

Um das Schreiben von Regeln zu vereinfachen, dürfen Stencils in Rollen zusammengefasst werden, die dann in den Regeln verwendet werden. So lassen sich in BPMN beispielsweise alle Start-Events in eine Rolle zusammenfassen und es muss nur einmal für die Rolle definiert werden, dass in ihnen kein Sequenzfluss enden darf.[Polo7]

Stencil Set Extension

Eine `StencilSetExtension` dient dazu, ein bestehendes `StencilSet` zu erweitern.

Der Aufbau gleicht dabei der des `StencilSets`, wobei es einige Besonderheiten zu beachten gibt. Mittels des Schlüsselwort `extends` kann ein zugrundeliegendes `StencilSet` definiert werden, auf dem die Erweiterung aufbaut.

In der Erweiterung können neue Stencils und Regeln definiert werden.

Existierende Stencils können in der Erweiterung nicht verändert werden. Es ist allerdings möglich, Stencils des zu erweiternden `StencilSets` aus der Erweiterung wegzulassen. Dazu wird das Schlüsselwort `removestencils` verwendet.

Es ist möglich, ein `StencilSet` mit mehreren `Extensions` gleichzeitig zu nutzen.

Plugins

Client-Plugins werden in JavaScript implementiert und mit dem Editor geladen. Dabei können sogenannte Profile definiert werden, die bestimmen, welche Art von Diagramm welche `StencilSets`, `StencilSetExtensions` und `Plugins` verwendet.

Plugins können über eine Fassade auf Funktionen von Oryx zugreifen.

Plugins haben mehrere Möglichkeiten, den Editor zu erweitern. Zum einen kann mittels `offer` dem Editor ein neues Kommando hinzugefügt werden, das dann als neues Icon in der Werkzeugleiste verfügbar gemacht wird. Zum anderen kann das Plugin auch eigene Bedienelemente wie beispielsweise eine Seitenleiste zum Editor hinzufügen. Einige Bedienelemente von Oryx sind so beispielsweise bereits als `Plugins` implementiert.[Tsc07]

Plugins können miteinander über Events kommunizieren. Ein Event hat einen Typ und kann darüber hinaus beliebige Daten enthalten. Events können mittels `raiseEvent` erzeugt werden. Ein Plugin kann sich für relevante Events als Listener registrieren. Tritt ein relevantes Event auf, wird eine entsprechende Callback-Funktion aufgerufen.[WB10]

4. Vorhandene Ansätze zur Variabilität

Dieses Kapitel bietet einen Überblick über vorhandene Ansätze zur Behandlung von Variabilität in Prozessen und zur Prozesskonfiguration.

Das Problem der Variabilität eines Prozesses kann in zwei Unterprobleme aufgeteilt werden. Zum einen muss die Variabilität modelliert werden, d.h. die Notation muss um geeignete Elemente erweitert werden, die Variabilität ausdrücken können. Zum anderen muss ein variabler Geschäftsprozess zur Ausführung gebracht werden. Dies kann man entweder erreichen, indem man ihn mittels Ableitung einer Variante in einen konkreten Prozess umwandelt oder, indem man die Ausführung des variablen Prozesses ermöglicht, wobei in diesem Fall die Variante zur Laufzeit abgeleitet wird.

Im Folgenden werden drei bisherige Ansätze zur Variabilität vorgestellt. Zum Teil decken diese beide Unterprobleme ab, zum Teil wird nur eines der beiden Teilprobleme behandelt.

Ein wichtiger Unterpunkt der Modellierung von Variabilität ist die Abhängigkeit zwischen einzelnen Teilen der Variabilität. So ergibt z.B. die Angabe der Höhe eines Prüfbetrags für eine Rechnung nur dann einen Sinn, wenn solch eine Prüfung auch ausgeführt werden soll. Eine weitere Abhängigkeit ist die Ausschließlichkeit von Optionen. So darf z.B. eine Rechnung nicht gleichzeitig per Nachnahme und Lastschrift beglichen werden. Sofern vorhanden, wird für die einzelnen Lösungen die Modellierung dieser Abhängigkeiten behandelt.

4.1. Modellierung von Variabilität mittels vorhandener Konstrukte

Die wohl einfachste Möglichkeit, Variabilität zu verwenden, ist es, sie mit den in BPMN vorhandenen Gestaltungselementen zu modellieren. So kann beispielsweise mittels eines Gateways eine Aktivität übersprungen werden, die je nach gewählter Variante stattfindet oder nicht. Bedingungen für die einzelnen Zweige sind dann die jeweiligen Ableitungsmöglichkeiten des Referenzprozesses.

Obwohl dieses Vorgehen prinzipiell zum Ziel führt, hat es einen entschiedenen Nachteil: Die Trennung der einzelnen Gestaltungsebenen wird nicht eingehalten. Für den Modellierer ist später nicht mehr erkennbar, ob es sich bei einer Verzweigung um Variabilität handelt, die einzelne Varianten voneinander unterscheidet oder um eine gewöhnliche Verzweigung, die sich in jeder Prozessinstanz ändert[HBRo8b].

Auch eine automatische Weiterverarbeitung der Variabilität im Sinne einer automatischen oder nutzerassistierten Variantenableitung ist durch diese Vermischung nicht mehr möglich, was ein weiteres Argument gegen eine Modellierung von Variabilität mit Bordmitteln ist.

4.2. Provop

Provop (*PRO*zessVarianten mittels *OPT*ionen) ist ein Modellierungsansatz der Universität Ulm. Der Grundgedanke von Provop ist es, einen Basisprozess zu modellieren, der mittels einer oder mehrerer sogenannter Optionen zu einem konkreten Prozess abgeleitet wird[HBRo8b].

Ein solcher Basisprozess muss kein existierender Prozess sein, sondern kann auch die Schnittmenge aller Varianten darstellen. Einzige Bedingung ist, dass er die Grundlage darstellt, auf der mittels Optionen neue Prozesse gebildet werden. Da eine Option auch die Löschung von Elementen beinhalten kann, muss der Basisprozess kein Minimalprozess sein[HBRo8b]. Ein Basisprozess enthält keinerlei explizite Erweiterungen, die über BPMN hinausgehen. Diese sind alle in den Optionen enthalten.

Eine Option besteht aus einem Optionsnamen und einer Menge von Anpassungen am Basisprozess[HBRo8b]. Diese Anpassungen können entweder Hinzufügungen (*INSERT*), Löschungen (*DELETE*), Verschiebungen (*MOVE*) oder Modifikationen (*MODIFY*) sein.

Die einzelnen Elemente der Diagramme werden in einem Repository verwaltet, das durch die einzelnen Optionen zusätzliche Elemente aufnehmen oder verwerfen kann. Grafisch stellt sich dies beispielsweise bei einer *INSERT*-Option als Prozessfragment dar.

Um den Ort der Optionsanwendung zu bestimmen, werden sogenannte Aufsetzpunkte verwendet. Diese Aufsetzpunkte definieren eindeutig einen Punkt im Prozess (z.B. vor oder nach einer Aktivität). Versieht man ein Prozessfragment mit Aufsetzpunkten, kann man es eindeutig in den Grundprozess einfügen.

Abbildung 4.1 zeigt ein Beispiel für eine Optionsanwendung. Mittels einer *INSERT*-Option wird an den Aufsetzpunkten vor der Aktivität „Übergabe“ eine Aktivität „Prüfung“ eingefügt.

Im Folgenden wird das Vorgehen bei den einzelnen Operationen genauer beschrieben.

INSERT Bei einem *INSERT* wird ein Prozessfragment aus dem Repository anhand zweier Aufsetzpunkte (Start- und Endpunkt) in das abzuleitende Modell eingefügt. Dabei ist zu beachten, wo diese Einfügung stattfindet. Liegen beide Aufsetzpunkte an den Enden eines Gateways, so wird das Prozessstück parallel zu den einzelnen Verzweigungen des Gateways eingefügt.

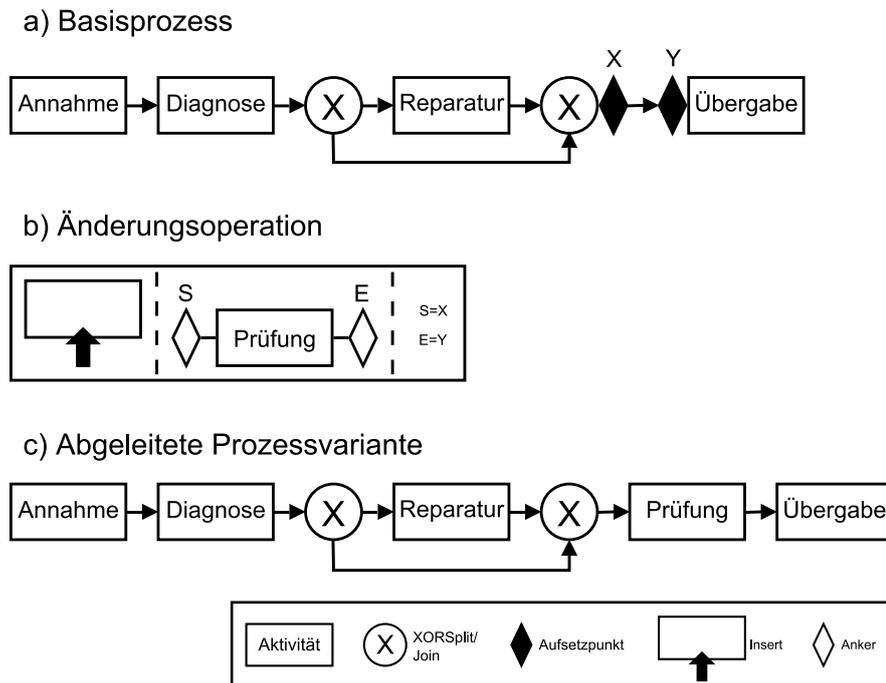


Abbildung 4.1.: Anwendung einer Option in Provop[Mau09]

DELETE Die *DELETE*-Operation unterscheidet zwei Arten, wie zu löschende Elemente spezifiziert werden. Zum einen können, ähnlich wie beim *INSERT*, Aufsetzpunkte verwendet werden, um Start und Ende des zu löschenden Abschnitts zu markieren. Zum anderen können zu löschende Elemente explizit benannt werden. Ersteres hat den Nachteil, dass durch vorherige Optionen eventuell etwas in den zu löschenden Bereich eingefügt wurde. Nach dem Entfernen der Elemente werden Prozesslücken (z.B. fehlender Sequenzfluss) automatisch durch Provop geschlossen.

MOVE Wie bei der ersten Variante von *DELETE* wird das zu verschiebende Prozessstück mittels Aufsetzpunkten definiert. Ein zweiter Satz von Aufsetzpunkten gibt an, wohin es verschoben werden soll. Dabei ist zu beachten, dass beide Sätze dieselbe Anzahl an Aufsetzpunkten haben.

MODIFY Mittels ID und Attributname erlaubt es die *MODIFY*-Operation, ein beliebiges Attribut eines Elements zu verändern. Beispiele dafür können Verzweigungs- und Schleifenbedingungen sowie Bearbeiterrollen sein. Bestimmte Attribute wie beispielsweise IDs können allerdings nicht geändert werden. In Provop werden sie als *fixe* Attribute bezeichnet.

Um aus dem Basisprozess eine konkrete Variante abzuleiten, wird eine Reihe von Optionen auf ihn angewandt. Dabei ergeben sich allerdings einige Probleme. So können sich Widersprüche ergeben, wenn z.B. eine Option einen Aufsatzpunkt hat, der durch eine vorherige Option gelöscht wurde. Außerdem ist es möglich, dass Optionen nicht kommutativ sind, d.h. dass je nach Anwendungsreihenfolge verschiedene Prozesse entstehen [HBRo8b]. In [HBRo8a] finden sich einige Beispiele für solche Ableitungsfehler.

Um diese Probleme abzuschwächen, erlaubt Provop es, Abhängigkeiten zwischen Optionen zu modellieren. Provop unterscheidet hier drei Konstrukte [HBRo9]:

Implication Mittels einer *Implication* lässt sich eine Abhängigkeit zwischen zwei Optionen angeben. Wenn Option A Option B impliziert, so muss bei der Anwendung von Option A vorher Option B angewendet worden sein.

Mutual Exclusion Mittels *Mutual Exclusion* lassen sich zwei Optionen gegenseitig ausschließen, sodass höchstens eine von ihnen angewandt werden kann.

Hierarchy Mit diesem Konstrukt lassen sich Optionen in einer Hierarchie anordnen. Wird eine Option aus der Hierarchie angewandt, werden vorher automatisch alle übergeordneten Optionen angewandt. Wird beispielsweise der Rechnungsversand per Einschreiben ausgewählt, so kann mittels Hierarchie automatisch die Option zur Einfügung der Rechnungserstellung angewandt werden.

Provop erlaubt die grafische Modellierung dieser Beziehungen. Sie werden bei der Ableitung einer Variante überprüft. Ist eine Bedingung verletzt, kann die Ableitung nicht durchgeführt werden [HBRo9].

Eine Modellierungsumgebung für Provop wurde unter dem Namen *Provop Demonstrator* als Erweiterung für Aris Business Architect umgesetzt [RRHB09]. Neben der Modellierung wird auch die Ableitung einer Variante mittels Optionen ermöglicht.

4.3. PESOA

PESOA ist ein durch das Bundesministerium für Bildung und Forschung gefördertes Gemeinschaftsprojekt unter der Federführung der Universität Potsdam, dessen Ziel es ist, eine Plattform für variantenreiche Prozesse zu implementieren [BGL⁺07]. Dazu wird der Ansatz des Product Family Engineering auf die Entwicklung von Prozessen zu übertragen. Product Family Engineering ist ein Ansatz zur Entwicklung von Softwarevarianten, bei dem basierend auf Kernkomponenten ein Reihe verwandter Softwareprodukte erstellt wird. [Noro8] Ein Vorteil dieses Ansatzes ist die Wiederverwendung dieser generischen Kernkomponenten in den verschiedenen Varianten.

PESOA will diesen Ansatz auch bei der Prozesserstellung nutzen und bezeichnet dies in Anlehnung an Product Family Engineering als *Process Family Engineering*. Dafür wird in [PSWW05] eine Klassifizierung von Mechanismen zur Variabilität vorgenommen. Im zweiten Teil der Arbeit werden diese Mechanismen für vorhandene Prozessmodellierungssprachen umgesetzt, soweit sie noch nicht vorhanden sind. Eine dieser Sprachen ist die BPMN, in der von Haus aus keiner der Mechanismen vorhanden ist [PSWW05]. Deswegen wird eine Erweiterung für BPMN vorgestellt, die ähnlich wie die UML auf sogenannten *Stereotypes* basiert. Stereotypes sind ein leichtgewichtiger Ansatz zur Erweiterung einer Notation, der es erlaubt, neue Modellelemente von existierenden abzuleiten, indem durch den Stereotype ein Verwendungszweck deklariert wird. Ein Stereotype kann außerdem durch sogenannte *tagged values* erweitert werden, die Stereotype-spezifische Attribute darstellen [PP05].

Zur Umsetzung der Variabilität wird BPMN mit einem Stereotype für einen Variation Point namens «*VarPoint*» erweitert. Dieser Stereotype hat zwei *tagged values*, nämlich *feature* und *type*. Mittels *type* wird festgelegt, welcher Typ von Variabilität in dem Variation Point zur Anwendung kommt. Um die Darstellung zu vereinfachen, kann dieser Typ auch anstelle von *VarPoint* direkt in die spitzen Klammern des Stereotype geschrieben werden. Die möglichen Typen sind *Abstract*, *Optional*, *Null* und *Default*. Sie werden im Folgenden näher beschrieben.

Mittels «*Abstract*» lässt sich eine Aktivität als abstrakte Basisaktivität kennzeichnen. Ähnlich wie eine abstrakte Klasse stellt sie keine konkrete Aktivität dar, sondern ist nur ein Platzhalter für eine von mehreren Alternativen. Mittels einer Assoziation können dann Subprozesse als konkrete Implementierungen mit der Basisaktivität verbunden werden. Diese Assoziation wird mit dem Stereotype «*Implementation*» versehen, um die Art der Beziehung kenntlich zu machen. Stehen mehrere Subprozesse zur Auswahl, kann einer von ihnen mittels «*default*» als voreingestellte Alternative gekennzeichnet werden. Ein solcher Subprozess wird mittels des Stereotype «*Variant*» als Variante gekennzeichnet. Grafisch wird dieser Stereotype durch ein stilisiertes Puzzleteil dargestellt.

Ein Beispiel für solch einen abstrakten Basisprozess findet sich in Abbildung 4.2. Hier kann eine Aktivität zur Bezahlung entweder durch eine Zahlung per Kreditkarte oder durch eine Wahlmöglichkeit zwischen Kreditkartenzahlung und Zahlung auf Rechnung ersetzt werden.

Will man hingegen, dass am Variation Point eventuell keine Aktivität stattfindet, kann man anstatt «*Abstract*» den Typ «*Null*» verwenden. Dieser gibt an, dass die Basisaktivität durch eine Variante ausgefüllt werden oder auch weggelassen werden kann. Eine solche Aktivität wird als *Extension Point* bezeichnet. Analog zu «*Abstract*» werden auch hier die konkreten Varianten mittels einer Assoziation hinzugefügt. Allerdings erhält diese hier den Stereotype «*Extension*», um die Optionalität zu verdeutlichen.

Gibt es nur eine mögliche Extension, so kann diese mit dem Variation Point zusammengefasst werden. Dazu wird der Stereotype «*Optional*» verwendet, mittels dessen sich eine Aktivität

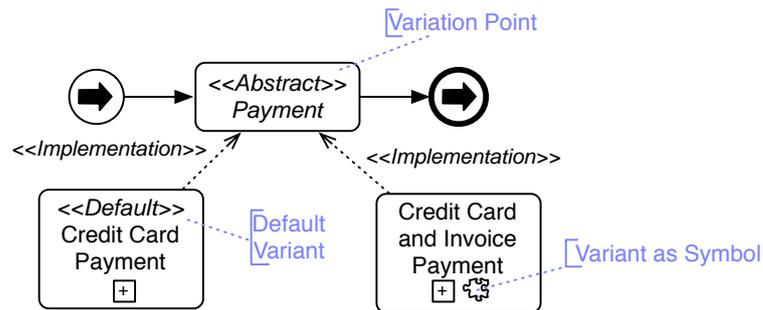


Abbildung 4.2.: Abstrakte Basisaktivität und zwei mögliche Implementierungen in PESOA.[PSWW05]

als optional kennzeichnen lässt, das heißt, sie kann in der abgeleiteten Variante vorhanden sein oder weggelassen werden.

Neben diesen Mechanismen zur Variabilität verfügt die Erweiterung außerdem über die Möglichkeit, BPMN-Attribute variabel zu definieren. Dazu wird das Attribut mittels einer *Group* isoliert. An diese *Group* werden dann Datenobjekte angehängt, in denen die alternativen Parameter stehen. Die verbindende Kante wird mit dem Stereotyp «*Parameterization*» gekennzeichnet, um die Variabilität kenntlich zu machen.

Außerdem skizziert die Erweiterung die Möglichkeit der Spezialisierung von Subprozessen mittels Vererbung. Dazu wird ein existierender Subprozess mittels «*VarPoint*» als variabel gekennzeichnet. Spezialisierte Subprozesse können dann wie bei «*Abstract*» mittels Assoziation als Variante hinzugefügt werden, die mit dem Stereotyp «*Inheritance*» versehen wird, um die Vererbung zu kennzeichnen. In der Erweiterung fehlt allerdings eine Beschreibung, wie solch ein Subprozess aufgebaut sein muss, damit er als spezialisierter Subprozess gilt[PSWW05].

Das zweite *tagged value, feature*, dient der Gruppierung von Variability Points. Alle Variability Points, die mit demselben *feature* versehen sind, müssen gemeinsam ausgewählt werden oder nicht. Ein Beispiel dafür ist eine gegenseitige Bewertungsfunktion am Ende eines Kaufvorgangs. Die Aktivität „Bewerten“ wird in diesem Fall sowohl beim Kunden als auch beim Anbieter mittels «*Optional*» in den Workflow eingefügt. Bei beiden ist als Feature „Bewertungsfunktion“ angegeben. So wird sichergestellt, dass in einer abgeleiteten Variante die Bewertung entweder bei beiden Teilnehmern oder bei keinem von beiden im Prozess enthalten ist.

Im Rahmen des PESOA-Prozesses wurde ein Prototyp zur Modellierung dieser Erweiterung entwickelt[BGL⁺07].

4.4. Prozesskonfigurator

Das Prozesskonfigurator-Projekt ist eine Sammlung von Werkzeugen für die Modellierung von Business Process Families, die ähnlich wie in PESOA als um Variabilität erweiterte BPMN-Prozesse modelliert werden[Wero8].

Der Ansatz trennt hierbei die Modellierung in drei Bereiche auf: Den Prozessfamilienkontext, die Prozessfamilienarchitektur und das variantenreiche Prozessmodell. Die ersten beiden Teile dienen der Abgrenzung von Problem- und Lösungsbeschreibung, ähnlich wie in der klassischen Softwareentwicklung die Unterscheidung von Spezifikation und Entwurf. Resultat der beiden Schritte sind ein Merkmals- bzw. ein Familienmodell, welche beide in der eigentlichen Prozesskonfigurator-Software erstellt werden[WMog].

Der Prozessfamilienkontext beschreibt die relevanten Einflussgrößen der Prozessfamilie als Merkmale. Hierfür werden diese Merkmale hierarchisch gegliedert. An oberster Stelle stehen die einzelnen Aspekte der Variabilität, die Merkmalkategorien und Merkmale enthalten. Ähnliche Merkmale werden in einer Merkmalkategorie zusammengefasst, wobei drei verschiedene Arten von Merkmalen unterschieden werden:

- Diskrete Merkmale enthalten als Ausprägungen Alternativen, von denen eine gewählt werden muss, ähnlich den abstrakten Aktivitäten und ihren Implementierungen in PESOA.
- Optionale Merkmale können entweder vorhanden sein oder nicht, analog zu dem Sterotype «*Optional*» in PESOA.
- Numerische Merkmale können einen Wert innerhalb eines Wertebereichs annehmen.

Diese hierarchische Gliederung wird im Prozesskonfigurator als Baum dargestellt.

Zur Modellierung von Beziehungen zwischen den einzelnen Merkmalen erlaubt der Prozesskonfigurator die Erstellung von Regeln, die als prädikatenlogische Formeln verfasst werden[Wero8]. So lässt sich beispielsweise die Abhängigkeit eines Merkmals von der Wahl eines anderen Merkmals oder der gegenseitige Ausschluss zweier Merkmale abbilden.

Auf die Modellierung des Familienkontexts folgt die Modellierung der Prozessfamilienarchitektur. Hier wird die Variabilität nicht in Aspekte und Merkmale, sondern in Prozessbausteine und Konfigurationsparameter unterteilt. In diesem Schritt geht es nicht um die Modellierung der Variabilität aus Anwendersicht, sondern aus Entwicklersicht in Bezug auf die konkrete Gestaltung des variablen Prozesses.

Prozessbausteine können hierarchisch untereinander gegliedert sein und ein oder mehrere Konfigurationsparameter enthalten. Ein solcher Konfigurationsparameter enthält mehrere Ausprägungen, von denen in der konkreten Variante genau eine gewählt werden muss. Mittels dieser Parameter werden die einzelnen Merkmale implementiert. Ein optionales

4. Vorhandene Ansätze zur Variabilität

Merkmal lässt sich beispielsweise mittels eines Konfigurationsparameters modellieren, der zwei Ausprägungen, nämlich *TRUE* und *FALSE* enthält.

Auch für die Prozessfamilienarchitektur können Regeln definiert werden, die Abhängigkeiten zwischen den einzelnen Prozessarchitekturelementen und zwischen Prozessarchitektur- und Prozesskontextelementen beschreiben. So kann eine Verbindung zwischen beiden Modellierungsebenen geschaffen werden.

Im dritten Schritt wird schließlich das variantenreiche Prozessmodell modelliert. Zur Modellierung wird ein handelsübliches BPMN-Werkzeug (STP BPMN Modeller) verwendet, da die BPMN für die Modellierung nicht erweitert wird. Statt zusätzliche Sprachelemente für Variabilität einzuführen, verwendet man Annotationen, in denen man Variation Points mittels einer eigenen Sprache beschreibt[WM09]. Ein Variation Point gibt eine Variable und eine zugehörige Bedingung an ihre Ableitung vor, die erfüllt sein muss, damit das entsprechende BPMN-Konstrukt im abgeleiteten Modell enthalten ist. Die Namen der Variablen entstammen dabei den Konfigurationsparametern des Familienmodells.

Sind alle drei Modelle erstellt, kann eine konkrete Variante abgeleitet werden. Hierzu werden die drei erstellten Modelle herangezogen. Im ersten Schritt werden im Familienkontext die gewünschten Merkmale ausgewählt. Dazu wird der Prozesskonfigurator verwendet. Im zweiten Schritt wird darauf aufbauend die Prozessfamilienarchitektur abgeleitet. Gewählte Merkmale werden anhand der angegebenen Regeln auf die Prozessfamilienarchitektur angewendet, sodass nur noch Entscheidungen getroffen werden müssen, die durch die Merkmale und die vorhandenen Regeln nicht schon feststehen.

Das Ergebnis dieser beiden Arbeitsschritte ist ein sogenanntes Variantenbeschreibungmodell in XML, das alle getroffenen Entscheidungen beinhaltet. Mittels dieses Modells und dem generischen BPMN-Prozess kann nun ein konkreter Prozess erstellt werden. Dazu dient eine Prozessmodell-Transformationskomponente.

4.5. Vergleich der verschiedenen Modellierungsansätze

Abschließend sollen die vorgestellten Ansätze in Bezug auf für die Aufgabenstellung relevante Kriterien verglichen werden. Tabelle 4.1 zeigt eine Übersicht dieses Vergleichs.

Da in einem unternehmensübergreifenden Prozess mehrere Parteien am Prozess beteiligt sind, muss auch die Variabilität eine Zuständigkeitsmodellierung unterstützen. Diese ist notwendig, um festzulegen, wer welche Entscheidungen bei der Variantenbildung trifft. Diese Unterstützung ist allerdings in keinem der Modellierungsansätze zu finden. Lediglich beim Prozesskonfigurator findet eine Trennung von Problem- und Lösungsbeschreibung statt, über die eine Trennung von Ableitung der Geschäftslogik und der Implementierungslogik denkbar ist. Die verschiedenen Sichten machen dies zumindest möglich.

4.5. Vergleich der verschiedenen Modellierungsansätze

Kriterium	Provop	PESOA	Prozesskonfigurator
Zuständigkeitsmodellierung	Keine	Keine	Trennung von Problem- und Lösungsbeschreibung
Abhängigkeitsmodellierung	Implication, Mutual Exclusion & Hierarchy	Features	Prädikatenlogik
Abhängigkeitsdarstellung	Separat	Integriert	Separat
Variantendarstellung	Separat mittels Optionen	Integriert mittels BPMN 2.0-Erweiterungen	Integriert mittels Annotationen
Implementierung	Provop Demonstrator[RRHB09]	Process Family Architect[WWWt]	Prozesskonfigurator[WWWu]

Tabelle 4.1.: Vergleich der Modellierungsansätze zur Variabilität

In Bezug auf die Modellierung der Abhängigkeiten finden sich bei den drei Ansätzen klare Unterschiede in Ausdrucksmächtigkeit und Komplexität. Der einfachste Ansatz sind die Features in PESOA, die lediglich die Abbildung von Genau-Dann-Wenn-Beziehungen ermöglichen. Provop bietet mit den drei genannten Konstrukten mehr Möglichkeiten zur Abhängigkeitsmodellierung, was allerdings auch zu einer höheren Komplexität führt. Im Prozesskonfigurator lassen sich mittels Prädikatenlogik wohl alle sinnvollen Abhängigkeitsbeziehungen ausdrücken, allerdings ist die Komplexität sowohl bei der Modellierung als auch bei der Implementierung des Werkzeugs am höchsten.

Die Darstellung der Abhängigkeiten kann auf verschiedene Arten erfolgen. Einerseits können sie direkt im Prozess abgebildet werden, wie es bei den Features in PESOA der Fall ist. Der Prozesskonfigurator hingegen enthält ein separates Werkzeug für die Abhängigkeitsmodellierung. In Provop werden Abhängigkeitsbeziehungen zwischen einzelnen Optionen in einem eigenen Diagrammtyp modelliert[HBR09].

Ähnlich wie bei den Abhängigkeiten kann auch bei der Variantendarstellung unterschieden werden, ob diese intern oder extern geschieht. Dies ist relevant, weil dadurch bestimmt wird, wie der variable Referenzprozess auszusehen hat. Provop ist hier am flexibelsten, denn mittels der Optionen werden die Varianten separat modelliert. Da eine Option allerdings auch eine Löschung von Elementen sein kann, hat man bei der Gestaltung des Referenzprozesses maximale Freiheit. Provop selbst schlägt hier vier verschiedene Ansätze zur Modellierung vor (z.B. als Minimalprozess oder Standardprozess)[HBR08b].

Bei den anderen beiden Ansätzen werden alle Bausteine für die einzelnen Varianten im selben Diagramm wie der Referenzprozess aufgeführt. Dies hat einerseits den Vorteil, dass eine

4. Vorhandene Ansätze zur Variabilität

Optionsverwaltung wie bei Provop entfällt, andererseits den Nachteil, dass das resultierende Diagramm unübersichtlich werden kann und einen roten Faden vermissen lässt.

Für alle drei Ansätze sind prototypische Umsetzungen vorhanden, allerdings ist zum Zeitpunkt der Recherche lediglich der Prozesskonfigurator online zu finden. Der Process Family Architekt wird nicht mehr auf der Projektseite erwähnt, eine Referenz findet sich nur noch auf einer archivierten Webseite.

5. Variabilitätskonzept

Aufbauend auf den im letzten Kapitel vorgestellten Ansätzen soll in diesem Kapitel ein Variabilitätskonzept erarbeitet werden, das die in Kapitel 4 vorgestellten Teilprobleme abdeckt und die in Kapitel 2 beschriebenen Anforderungen erfüllt. Dazu wird das Konzept in drei Teilgebiete aufgeteilt: Variabilitätsmodellierung, Abhängigkeitsmodellierung und Variantenableitung.

5.1. Variabilitätsmodellierung

5.1.1. Variabilitätsumfang

Für die Variabilitätsmodellierung ist zuerst eine Entscheidung zu treffen, welche Art von Variabilität notwendig bzw. wünschenswert ist. Gibt es nämlich zu wenig Variabilitätskonstrukte, lässt sich eventuell gewünschte Variabilität gar nicht oder nur mittels Umwegen umsetzen. Gibt es allerdings zu viele Variabilitätskonstrukte, steigt der Einarbeitungsaufwand für den Modellierer und sinkt die Lesbarkeit der variantenreichen Diagramme. Dies ist insbesondere deswegen von Bedeutung, da bei unternehmensübergreifenden Prozessen davon ausgegangen werden kann, dass auch Fachanwender mit dem Modell arbeiten. Deshalb sollte die Komplexitätssteigerung im Vergleich zu gewöhnlichem BPMN minimal gehalten werden.

Um zu untersuchen, welche Art von Variabilität in Bezug auf diese Punkte sinnvoll ist, werden die einzelnen Elemente der BPMN untersucht. Für eine Übersicht über die untersuchten Elemente empfiehlt sich [WWWb].

Aktivitäten

Unter Aktivitäten versteht man in BPMN unter Anderem Tasks und Subprozesse. Variabilität in Aktivitäten ist z.B. das Ersetzen eines abstrakten Tasks durch eine oder mehrere Varianten oder das Weglassen der Aktivität. Dies wird auch von allen drei Modellierungsansätzen ermöglicht.

Austauschbarkeit von Aktivitäten zu ermöglichen ist eines der wichtigsten Variabilitätskonzepte. Betrachtet man PESOA, so stellt man fest, dass außer variablen Attributen lediglich variable Aktivitäten zugelassen werden.

Davon zu unterscheiden ist die Variabilität innerhalb ein und desselben Tasks. So ist beispielsweise denkbar, dass man die Art des Tasks (Nachrichtenbehandlung, Script, Nutzeraktion, etc.) variabel ändern kann. Analog dazu könnte man auch die Art der Wiederholung des Tasks ändern (einmal, Schleife, etc.). Diese beiden Änderungen sind allerdings leicht durch den Austausch der Tasks umzusetzen. Außerdem führt eine solche Art von Variabilität zu einem schwer verständlichen Referenzprozess, da der Kontrollfluss bei variabler Wiederholung nicht ohne Weiteres ersichtlich ist.

Aus diesen Gründen wird auf die Variabilität dieser beiden Eigenschaften verzichtet.

Gateways

Variabilität in einem Gateway kann neben den Bedingungen für den ausgehenden Fluss auch die Art des Gateway selbst sein.

Die erste Art von Variabilität wird durch das Ändern von Attributen in allen vorgestellten Ansätzen ermöglicht und ist auch praxisrelevant. So lässt sich beispielsweise mittels der Gateway-Bedingung ein Grenzbetrag festlegen, ab dem eine manuelle Prüfung eines Auftrags vorgenommen werden muss.

Das Umwandeln verschiedener Gatewayarten ineinander, beispielsweise von exklusivem in paralleles Gateway und umgekehrt wird hingegen lediglich von Provop ermöglicht, indem man das Gateway mittels Option löscht und ein anderes einfügt.

Für solch eine Variabilität konnte in einer ersten Analyse der Praxisbeispiele kein Beispiel gefunden werden. Änderungen der Gatewayarten haben grundlegende Änderungen des Kontrollflusses zur Folge. So kann beispielsweise Nebenläufigkeit in einen Prozess eingeführt werden, in dem es vorher keine gab. Bei einem Event-basierten Gateway ändert sich außerdem die Semantik der nachfolgenden Events. Änderungen der Gateways ziehen also weitreichende Änderungen der Ausführungssemantik mit sich, die insbesondere für Nutzer ohne technischen Hintergrund schwer verständlich sind.[Siloga]

Sollte solch eine Konstruktion tatsächlich benötigt werden, kann sie auch mittels zweier alternativer Subprozesse modelliert werden. Variable Gateway-Arten werden folglich in dieser Arbeit nicht umgesetzt.

Ereignisse

Ereignisse sind in BPMN in vielerlei Hinsicht kategorisiert. Zum einen wird unterschieden, wann das Ereignis auftritt (Prozessstart, Prozessende, Subprozess-unterbrechend, etc.). Dies variabel zu gestalten ist wenig sinnvoll, da im Regelfall bei einer Umwandlung zwischen den Typen die BPMN-Syntax verletzt wird. Die zweite Kategorisierung ist nach Art des Ereignisses (Nachricht, Signal, Fehler, etc.). Auch hier ist eine Umwandlung in vielen Fällen problematisch, da verschiedene Ereignisse auf verschiedene Arten in einem Diagramm verwendet werden können. So darf z.B. nur in einem Nachrichtenergebnis ein Nachrichtenfluss enden. Aus diesem Grund wird diese Art von Variabilität ebenfalls nicht zugelassen.

Variable Ereignisse können allerdings in variablen Aktivitäten eingesetzt werden. So kann man z.B. in einem variablen Subprozess eigene Ereignisse verwenden.

Pools und Lanes

Eine Hinzu- oder Wegnahme von Pools und Lanes bedeutet, eine Rolle zum Prozess hinzuzufügen oder wegzunehmen. Keines der vorgestellten Modelle erwähnt explizit die Behandlung von Pools und Lanes, obwohl sie durch Provops Konzept umsetzbar ist.

Da eine solche Veränderung der Rollen einen tiefgreifenden Eingriff in den Prozess darstellt und auch im Anwendungsfall nicht vorkommt, wird diese Art von Variabilität nicht berücksichtigt. Muss man sie dennoch umsetzen, kann für jede Rollenkonstellation ein eigener Referenzprozess erstellt werden.

Attribute

Aus den Referenzprozessen des Praxisprojekts geben sich drei primäre Einsatzgebiete von variablen Attributen:

Grenzwerte Setzen von Grenzwerten, beispielsweise dem Betrag, ab dem ein bestimmter Zweig im Diagramm gewählt werden soll.

Gateway-Bedingungen Variable Gatewaybedingungen ermöglichen es, den Prozess anzupassen, ohne die Struktur zu verändern.

Implementierungsdetails Mittels variabler Attribute kann die Implementierung der einzelnen Teile des Geschäftsprozesses ermöglicht werden. Ein Beispiel für ein Implementierungsdetail ist das Binden eines Tasks an einen konkreten Webservice.

Darüber hinaus ist allerdings denkbar, dass in anderen Anwendungsfällen auch eine Variabilität anderer Attribute notwendig sein wird. Als Kompromiss zwischen den identifizierten Anwendungsfällen und einer größtmöglichen Flexibilität der Lösung wird es ermöglicht, jedes Freitext-Attribut mit einem Variabilitätspunkt zu versehen. Allerdings werden zunächst nur die identifizierten Attribute angeboten. Andere Attribute müssen in der fertigen Lösung explizit vom Nutzer angefordert werden, beispielsweise über eine Schaltfläche oder eine Expertenansicht. Es liegt dann am Modellierer des Referenzprozesses, dies sinnvoll einzusetzen.

Sequenzfluss

Durch das Einfügen und Austauschen von Aktivitäten ist zwangsläufig variabler Sequenzfluss vonnöten. Die Frage ist, inwieweit Sequenzfluss zwischen festen Aktivitäten variabel sein soll. So wäre z.B. denkbar, dass die Behandlung eines unterbrechenden Ereignisses variabel von mehreren Aktivitäten übernommen werden kann.

Der Nachteil an einer solchen Veränderung des Sequenzflusses ist, dass diese Art der Variabilität im Referenzprozess schwer erkennbar ist, da sie sich nicht in der Nachbarschaft von variablen Aktivitäten befinden muss. Deswegen ist es sinnvoll, variablen Sequenzfluss so einzuschränken, dass er nur in Verbindung mit variablen Aktivitäten auftreten kann. Wird wie im oben beschriebenen Fall variabler Fluss benötigt, muss dann eine der verbundenen Aktivitäten variabel gestaltet werden.

Nachrichtenfluss

Variabler Nachrichtenfluss umfasst die Quelle, das Ziel und den Inhalt der Nachricht. Da hinzugefügte Aktivitäten zusätzliche Nachrichten bzw. Nachrichteninhalte erfordern können, muss variabler Nachrichtenfluss ermöglicht werden. Wie beim Sequenzfluss wird er aber lediglich in Verbindung mit variablen Aktivitäten erlaubt.

Bei einer Veränderung von Quelle und Ziel ist zu bedenken, dass die Quelle bzw. das Ziel auch vorhanden sein müssen. Ein Nachrichtenfluss kann nämlich z.B. in variantenspezifischen Diagrammteilen beginnen oder enden. ProVop erlaubt dies beispielsweise durch Aufsetzpunkte.

Daten und Datenfluss

Datenfluss wird in BPMN ähnlich modelliert wie Nachrichtenfluss, nur, dass Datenfluss zwischen Aktivitäten eines Prozessteilnehmers verläuft. Damit gilt wie auch beim Nachrichtenfluss, dass neue Aktivitäten neuen Datenfluss nötig machen. Auch hier wird also variabler Datenfluss in Verbindung mit variablen Aktivitäten erlaubt.

5.1.2. Modellierungskonzept

Nachdem im vorausgehenden Abschnitt der Umfang der Variabilität definiert wurde, wird in diesem Abschnitt ein Konzept vorgestellt, das es ermöglicht, die gewünschte Variabilität umzusetzen.

In Kapitel 4 wurden bisherige Variabilitätsansätze vorgestellt und miteinander verglichen. Anhand der Vergleichskriterien lassen sich die wichtigsten Entwurfsentscheidungen ablesen, die bei einem Konzept zur Variabilitätsmodellierung getroffen werden müssen. Dabei spielen auch die in Kapitel 2 beschriebenen Anforderungen eine wichtige Rolle.

Grundsätzlich gibt es zwei Möglichkeiten, Variabilitätsmodellierung vorzunehmen. Entweder erweitert man die Modellierungssprache oder drückt Variabilität mit vorhandenen Mitteln aus. Eine Erweiterung besteht aus zusätzlichen Modellierungselementen wie beispielsweise den variablen Tasks in PESOA. Allerdings lässt sich auch ohne zusätzliche Elemente Variabilität modellieren, wie dies im Prozesskonfigurator mit Annotationen geschieht.

Wie bei der naiven Variabilitätsmodellierung mit Gateways etc. besteht auch bei der Umwidmung vorhandener Konstrukte zur Variabilitätsmodellierung das Problem, dass die Variabilität aus dem modellierten Diagramm nicht ohne Weiteres ersichtlich ist. Für einen geschulten Modellierer ist die Verwendung von Annotations oder speziellen Namen vielleicht ausreichend, aber da wie beschrieben davon auszugehen ist, dass das variantenreiche Modell auch von Fachanwendern verwendet wird, muss die Variabilität klarer zu erkennen sein.

Deswegen fällt die Entscheidung auf eine Erweiterung der Modellierungssprache BPMN um zusätzliche Konstrukte.

Des Weiteren stellt sich die Frage, ob variable Bausteine separat vom Referenzprozess oder integriert modelliert werden sollen. Für eine integrierte Modellierung spricht, dass alle Varianten in einem Dokument zusammengefasst sind. Ein Nachteil integrierter Modellierung ist wie beschrieben die mangelnde Übersicht im Referenzprozess, die sich mit zunehmender Variantenanzahl verschlechtert. Betrachtet man unternehmensübergreifende Prozesse, so ergibt sich ein weiterer Nachteil. Möchte man den einzelnen Nutzern der Plattform die Modellierung eigener Bausteine für Varianten ermöglichen, so muss man ihnen Zugriff auf den Referenzprozess ermöglichen. Dies ist bei einer realistischen Kundenzahl nicht praktikabel, da unter anderem eine nebenläufige Bearbeitung des Referenzprozesses vonnöten wäre.

Bei der separaten Modellierung variabler Bausteine hingegen ist die Darstellung des Referenzprozesses unabhängig von der Zahl variabler Bausteine, da diese erst bei der Ableitung der Variante sichtbar werden. Bedenkt man, dass diese Ableitung auch für Fachanwender möglich sein soll, so steht fest, dass der Referenzprozess auch für diese verständlich sein muss. Ein übersichtlicherer Referenzprozess und eventuell eine Reduzierung der angebotenen Bausteine trägt dazu bei.

Ein weiterer Vorteil der separaten Modellierung ist die Erfüllung der für *openXchange* definierten Sicherheitsanforderungen (siehe 2.2). Der Plattformbetreiber modelliert den Referenzprozess, auf den die Plattformteilnehmer nur Lesezugriff haben. Die Prozessteilnehmer können eigene Bausteine in separaten Diagrammen modellieren, auf die nur sie Zugriff haben.

Aus diesen Gründen fällt die Entscheidung auf separate Variantenmodellierung. Betrachtet man allerdings den Provop-Ansatz, so muss man feststellen, dass dieser zwar sehr mächtig, aber dadurch auch sehr komplex ist. Vertauschungen und Löschungen von Prozessteilen durch Varianten mögen dem Modellierer des Referenzprozesses zwar die Arbeit erleichtern, aber dem Fachanwender, der die Ableitung vornimmt, wird so das Verständnis der einzelnen Wahlmöglichkeiten erschwert. Hinzu kommt noch das Problem, dass die fehlende Kommutativität der Operationen sowohl bei der Referenzprozessmodellierung als auch bei der Variantenableitung eine weitere Fehlerquelle darstellt.

Deswegen erscheint es sinnvoll, die Veränderungsmöglichkeiten eines variablen Bausteins einzuschränken. Lässt man nur Hinzufügung von Elementen zu, so erhält man einen Referenzprozess, der die Schnittmenge aller Prozessvarianten [HBRo8b] darstellt, d.h. dass alle Elemente im Referenzprozess auch in jeder Variante enthalten sind.

Für die Ableitung einer Variante ergibt sich dann ein additives Vorgehen nach dem Baukastenprinzip, was für die Fachanwender gut verständlich ist.

Mit diesen Entwurfsentscheidungen stellt sich die Frage, wie die Variabilität im Referenzprozess verankert wird. Provop benutzt dazu Aufsetzpunkte, die Start und Ende eines einzusetzenden Prozessfragments angeben. Die anderen vorgestellten Ansätze haben keine separate Modellierung, aber betrachtet man PESOA, so ergibt sich eine weitere Möglichkeit.

PESOA bietet eine abstrakte Aktivität, in die bei der Ableitung eine Implementierung eingesetzt wird. Erweitert man dies auf separate Modellierung, so ergäbe sich eine Art Black Box im Sequenzfluss, in die bei der Variantenableitung ein Prozessbaustein eingesetzt wird.

Dieser Ansatz hat gegenüber dem von Provop den Vorteil, dass der Ort, an den ein Prozessfragment eingesetzt wird, explizit durch eine Leerstelle gekennzeichnet ist. Dies erleichtert dem Fachanwender die Übersicht, wo Ableitungsentscheidungen getroffen werden müssen und welche Verbindungen eingesetzte Prozessbausteine mit dem Referenzprozess haben.

Um variable Attribute zuzulassen, gibt es zwei Ansätze. Zum einen kann ein Attribut mittels einer zusätzlichen Wahlmöglichkeit als variabel deklariert werden (z.B. per CheckBox o.Ä.) und dann die Variabilität separat modelliert werden. Zum anderen kann die Variabilität im Attribut selbst beschrieben werden (z.B. mittels regulären Ausdrücken). Da eine Umsetzung der Variabilitätsmodellierung durch Oryx vorgesehen ist, wird der zweite Ansatz gewählt. Variabilität wird im Attribut mit einer eigenen Regelsprache definiert. Dies hat den Vorteil, dass dazu die BPMN nicht verändert werden muss. Ist dies für den Nutzer zu unkomfortabel,

lässt sich gegebenenfalls ein Wizard implementieren, der es ermöglicht, diese Sprache in einer grafischen Ansicht zu editieren.

Resultierend aus den getroffenen Entwurfsentscheidungen ergibt sich folgender Grobaufbau:

Variabler Referenzprozess Ein BPMN-Prozess, der Leerstellen für einzusetzende Prozessfragmente enthält. Außerdem können seine Elemente variable Attribute enthalten.

Prozessfragment Ein separat modellierter Teilprozess, der in einen Referenzprozess eingesetzt werden kann.

Variable Attribute Attribute, die mittels einer eigenen Beschreibungssprache definiert werden, die auf Variabilitätsbeschreibungen aufbaut.

Variantenableitung Auswahl der einzusetzenden Prozessfragmente in die Leerstellen im Referenzprozess durch den Nutzer. Auswahl konkreter Werte für die variablen Attribute. Als Resultat der Ableitung entsteht eine Prozessvariante.

Prozessvariante Ein abgeleiteter Referenzprozess, der keine Variabilität mehr enthält. Er ist ein gültiger BPMN-Prozess.

Um dies umzusetzen, wird im Folgenden eine zweiteilige Erweiterung von BPMN 2.0 definiert, die zum einen die Modellierung der Referenzprozesse und zum anderen die Modellierung der Prozessfragmente umfasst. Da für die Speicherung der variablen Prozesse Variabilitätsbeschreibungen (siehe 3.2) verwendet werden sollen, stellt diese Erweiterung eine reine Modellierungshilfe dar. Das BPMN 2.0-Metamodel wird nicht erweitert.

Variabler Referenzprozess

Um den Referenzprozess variabel zu gestalten, muss BPMN um ein neues Element erweitert werden, das als Platzhalter für einzufügende Prozessfragmente dient. Wir nennen dieses Element *Variable Region*.



Abbildung 5.1.: Darstellung einer variablen Region.

Dargestellt wird diese variable Region als abgerundetes Rechteck mit einem Puzzleteil.

Eine variable Region wird wie ein Task in den Kontrollfluss eingebunden, allerdings muss sie genau einen ein- und genau einen ausgehenden Sequenzfluss haben. Dies hat den Grund, dass sie durch beliebige BPMN-Elemente ersetzt werden kann und bei mehreren ein- und ausgehenden Kanten der Ablauf des Referenzprozesses unklar ist. Ein Beispiel dafür wären verschiedene Arten von Gateways in einem Prozessfragment: Bei mehreren ausgehenden Flüssen würde beispielsweise je nach Variante einer oder mehrere der Pfade parallel gewählt, ohne, dass dies aus dem Referenzprozess ersichtlich wäre.

Aus einem ähnlichen Grund dürfen in einer variablen Region keine Nachrichtenflüsse beginnen oder enden. Ein solcher Nachrichtenfluss müsste je nach gewähltem Fragment an ein spezifisches Element im Fragment anschließen. Zum einen ist dies a priori nicht ersichtlich, zum anderen ist nicht sichergestellt, dass das eingesetzte Fragment überhaupt ein passendes Element dafür enthält. Soll dennoch ein Fluss über Fragmentgrenzen hinaus modelliert werden, so kann dies im Prozessfragment erledigt werden.

Die variable Region hat folgende Attribute:

Name Name der variablen Region. Wird als Beschriftung angezeigt.

Documentation Kommentar.

Prozessfragmente

Ein Prozessfragment ist ein beliebiges Teilstück eines BPMN-Prozesses, das bei der Ableitung in den Referenzprozess eingesetzt wird. Es kann beliebige BPMN-Elemente enthalten außer Pools und Lanes, da durch ein Fragment keine neuen Rollen eingeführt werden dürfen. Außerdem könnten dadurch bei der Ableitung inkorrekte Prozesse entstehen.

Da bekannt ist, dass die korrespondierende variable Region im Referenzprozess genau einen ein- und einen ausgehenden Sequenzfluss hat, benötigt man im Prozessfragment zwei Aufsetzpunkte, ähnlich denen, die in Provop Verwendung finden. Anhand der Aufsetzpunkte wird dann das Prozessfragment in den Referenzprozess eingesetzt.

Ein naiver Ansatz wäre, für diese Punkte Prozessstart und -ende zu wählen. Dies hat aber zwei Nachteile. Zum einen ist dadurch die Semantik unklar, da es sich bei einem Prozessfragment eben nicht um einen Subprozess handelt, zum anderen kann ein Prozess mehrere Start- und Endevents enthalten sowie die einzelnen Events mehrere aus- bzw. eingehende Flüsse haben.

Deswegen werden für Fragment-Start und Fragment-Ende zwei neue Elemente eingeführt. Als Form für diese wurde ein Dreieck gewählt, weil es bisher in der BPMN noch keine Verwendung findet.

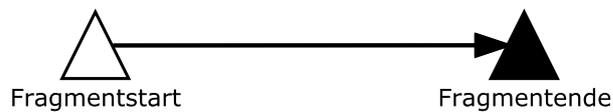


Abbildung 5.2.: Darstellung von Fragment-Start und -Ende.

Der Fragment-Start wird als weißes Dreieck modelliert, analog zur weißen Farbgebung von Start-Events. Er hat genau einen ausgehenden Sequenzfluss und darf keinerlei Nachrichtenfluss haben.

Das Fragment-Ende wird als schwarzes Dreieck modelliert, analog zur Gestaltung von End-Events als geschwärzte Start-Events. Es hat genau einen eingehenden Sequenzfluss und darf keinerlei Nachrichtenfluss haben.

Sowohl Fragment-Start als auch Fragment-Ende haben folgende Attribute:

Name Name des Elements. Wird als Beschriftung angezeigt.

Documentation Kommentar.

Jedes Prozessfragment muss genau einen Fragment-Start sowie genau ein Fragment-Ende enthalten, die sich beide auf der obersten Ebene des Diagramms befinden müssen, d.h. sie dürfen in keinem anderen Element (z.B. einem Subprozess) enthalten sein.

Ein Prozessfragment kann seinerseits auch wieder variable Regionen enthalten. Um diese Rekursion aufzulösen, wird bei der Ableitung zuerst aus dem variablen Prozessfragment eine Variante erstellt und diese dann in den Referenzprozess eingesetzt.

Möchte man Prozess- oder Nachrichtenfluss über die Grenzen eines Prozessfragments hinweg ermöglichen, steht dafür ein sogenannter Fragment-Link bereit, der als zwei ineinanderliegende Dreiecke dargestellt wird.

Ein solcher Fragment-Link dient ähnlich wie ein Link-Element der Verbindung von Elementen. In einem Fragment-Link kann maximal ein Sequenz- oder Nachrichtenfluss enden, der bei der Ableitung über die Fragment-Grenze hinaus ausgeweitet wird. Das Ziel des ausgehenden Flusses wird dabei als Attribut angegeben und kann somit ebenfalls variabel sein.

Da ein solcher Link auch in ein anderes Prozessfragment führen kann, ist es Aufgabe des Modellierers, sicherzustellen, dass das Ziel des Fragment-Links zum Zeitpunkt des Einsetzens bereits in der Variante vorhanden ist.

Der Fragment-Link hat folgende Attribute:

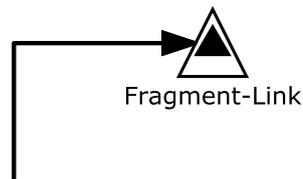


Abbildung 5.3.: Darstellung eines Fragment-Links.

Name Name des Fragment-Links. Wird als Beschriftung angezeigt.

Target Ziel des Fragment-Links.

Documentation Kommentar.

Abbildung 5.4 zeigt beispielhaft das Einsetzen eines Fragments mit einem Fragment-Link. Das Fragment wird in die variable Region des Referenzprozesses eingesetzt. Der Sequenzfluss der Region wird mit Fragment-Start und Fragment-Ende verbunden. Der Nachrichtenfluss, der am Fragment-Link endet, wird mit dem Ziel des Links im Referenzprozess verbunden.

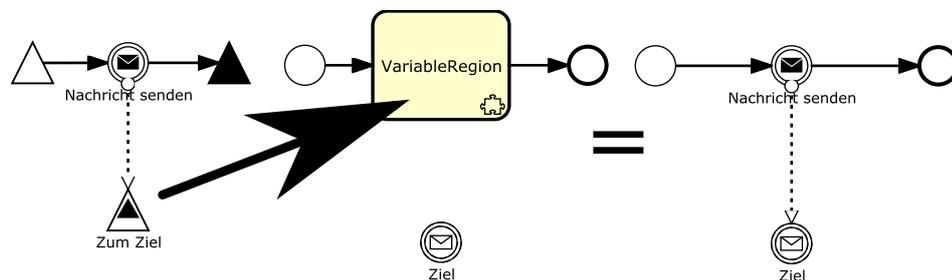


Abbildung 5.4.: Einsetzung eines Fragments mit einem Fragment-Link.

Variable Attribute

Für variable Attribute wird eine eigene Sprache definiert, mittels der ein Attribut als variabel erklärt werden kann. Ein Ausdruck in dieser Sprache ersetzt dann den eigentlichen Inhalt des Attributs.

Dazu ist zuallererst zu klären, welche Art von Variabilität für ein Attribut sinnvoll ist. Als Orientierung dazu dienen die in 3.2 vorgestellten Alternativen des Variability Descriptor.

Explizite Alternative Analog zur expliziten Alternative kann ein expliziter Wert als Wahlmöglichkeit für das Attribut vorgegeben werden.

Zeiger-Alternative Da zum Zeitpunkt der Referenzprozessmodellierung noch kein Dokument existiert, ist eine Zeigeralternative wie im Variability Descriptor nicht sinnvoll.

Leere Alternative Die leere Alternative ermöglicht es, das Attribut leerzulassen.

Ausdrucksalternative Wie bei der Zeigeralternative ist hier zu bedenken, dass noch kein zugrundeliegendes Dokument existiert. Denkbar wäre der Verweis auf ein Attribut eines anderen Elements. Dies wird allerdings vorerst nicht umgesetzt.

Freie Alternative Analog zur freien Alternative der Variabilitätsbeschreibung soll in den Attributen die Eingabe von Freitext ermöglicht werden, der allerdings auch mittels Constraints eingeschränkt werden kann.

Anhand dieser Anforderungen wurde eine Ausdruckssprache entwickelt, die durch folgende EBNF definiert ist. Die Syntax orientiert sich dabei an der des Variability Descriptor. Um die Implementierung zu erleichtern, gründet sich die Syntax auf der *JavaScript Object Notation*[WWWc], einem Datenaustauschformat, das eine Untermenge von JavaScript ist.

```
Attribute ::= <Variable_Attribute>|<Fixed_Attribute>

Variable_Attribute ::= \{"alternatives":<Alternative_List>\}

Fixed_Attribute ::= <String>

Alternatives_List ::= [<Alternative> {, <Alternative>}*]

Alternative ::= \{<Fixed_Alternative>\}|\{<Free_Alternative>\}

Fixed_Alternative ::= "type":"E","text": "<String>"

Free_Alternative ::= "type":"F","text": "<Constraint>"

<Constraint> ::= siehe Constraints des Variability Descriptor
```

Listing 5.1: Attributsprache

Mittels dieser Ausdruckssprache lassen sich eine Anzahl von expliziten Alternativen und freien Alternativen definieren. Die leere Alternative lässt sich abbilden, indem man eine explizite Alternative mit leerem Text erstellt.

5.1.3. Zusammenfassung

Im Rahmen einer Validierung wird die beschriebene Erweiterung eingesetzt, um den in Abschnitt 5.1.1 abgesteckten Variabilitätsumfang zu modellieren. Dafür soll in Tabelle 5.1 ein

5. Variabilitätskonzept

Überblick gegeben werden, in dem kurz aufgezeigt wird, wie sich entsprechende Variabilität modellieren lässt.

Element	Variabilität	Umsetzung
Aktivitäten	Austauschbarkeit	Verschiedene Prozessfragmente mit einzelnen Aktivitäten
Aktivitäten	Aktivitätsart und Wiederholung	keine (alternativ: mehrere Aktivitäten)
Gateways	Bedingungen	variables Attribut
Gateways	Typ des Gateways	keine (alternativ: mehrere Fragmente)
Ereignisse	Typ des Ereignisses	keine (alternativ: mehrere Ereignisse)
Pools und Lanes	Hinzufügen / löschen	keine (alternativ: mehrere Referenzprozesse)
Attribute	variabler Wert	variable Attribute
Sequenzfluss	Quelle/Ziel	Fragment-Link
Nachrichtenfluss	Inhalt	variables Attribut
Nachrichtenfluss	Quelle/Ziel	Fragment-Link
Datenfluss	Quelle/Ziel	Fragment-Link
sonstige Elemente	allgemein	keine

Tabelle 5.1.: Modellierbarkeit und Umsetzung von Variabilität

5.2. Abhängigkeitsmodellierung

Bei der Modellierung variabler Prozesse existieren Abhängigkeiten zwischen den einzelnen variablen Elementen. Beispiele für solche Abhängigkeiten sind, dass ein Fragment nur eingesetzt werden darf, nachdem ein bestimmtes anderes Fragment eingesetzt wurde, dass für zwei unabhängige Prüfungen nicht derselbe Prüfer ausgewählt werden darf, oder, dass ein bestimmter Prozessschritt nur einmal in einem Prozess durchgeführt werden darf.

Diese Abhängigkeiten sollen nicht nur implizit vorhanden sein, sondern sie sollen auch bei der Ableitung erzwungen werden. Deswegen müssen sie explizit modelliert werden.

Im Zuge der Abhängigkeitsmodellierung werden einzelne variable Elemente als *Variabilitätspunkte* betrachtet, die konzeptuell den Variability Points der Variabilitätsbeschreibungen (siehe 3.2) entsprechen. In der vorgestellten Erweiterung sind Variabilitätspunkte entweder variable Regionen oder variable Attribute.

Wie auch bei der Variabilität selbst bietet der Vergleich der bisherigen Ansätze (siehe 4.5) Anhaltspunkte, welche Entwurfsentscheidungen bei der Abhängigkeitsmodellierung zu treffen sind.

Bevor man sich Gedanken machen kann, wie Abhängigkeiten modelliert werden können, muss geklärt werden, in welchem Umfang Abhängigkeiten zugelassen werden. Alle drei vorgestellten Ansätze unterscheiden sich in der Mächtigkeit ihrer Modellierungsansätze. Da ein Ziel der Arbeit ist, Variabilitätsbeschreibungen zu verwenden, sind deren Fähigkeiten zur Abhängigkeitsbeschreibung maßgebend. Führt man darüber hinaus zusätzliche Konstrukte ein, so müssen sich diese bei der Umwandlung in eine Variabilitätsbeschreibung so übersetzen lassen, dass sie mit den vorhandenen Mitteln ausgedrückt werden.

Ebenso wie Fragmente können Abhängigkeiten separat modelliert werden, wie es zum Beispiel in Provop geschieht. Dies schafft eine Übersicht über das Abhängigkeitsgeflecht zwischen den einzelnen Variabilitätspunkten, sodass sich beispielsweise zyklische Abhängigkeiten erkennen lassen. Der Nachteil ist allerdings, dass durch die separate Modellierung nicht mehr klar ersichtlich ist, wie die Elemente im Sequenzfluss zueinander angeordnet sind, da sich diese Informationen nicht im Abhängigkeitsdiagramm befinden. Für den Fachanwender hat diese Entscheidung keine Auswirkungen, da die Abhängigkeiten bei der Variantenableitung automatisch aufgelöst werden.

Für Variabilitätsbeschreibungen existiert bereits eine solche grafische Notation[Dino8], die allerdings der Modellierung einer gesamten Beschreibung und damit eines gesamten Prozesses dient. Deswegen ist sie für diese Arbeit ungeeignet, da die Fragmente separat modelliert werden sollen. Um sie einzusetzen, müssten entweder separate Diagramme für Referenzprozess und Fragmente modelliert werden, die dann verknüpft würden oder die Abhängigkeiten von Referenzprozess und den zugehörigen Fragmenten in einem einzelnen Diagramm modelliert werden. Ersteres bedeutet zusätzlichen Aufwand bei der Modellierung, letzteres läuft dem Konzept der separaten Fragmentmodellierung zuwider.

Denkbar ist auch eine Modellierung wie im Prozesskonfigurator, die die Abhängigkeiten in zwei Stufen unterteilt, um dem Fachanwender die Ableitungsentscheidungen zu erleichtern. Genau diese Möglichkeiten bieten allerdings auch die Variabilitätsbeschreibungen, aus denen sich mittels eines Tools ein Wizard generieren lässt, mit dem man eine Variante ableiten kann.[Aueo8]

Da diese Möglichkeit also bereits durch den Export in eine Variabilitätsbeschreibung abgedeckt ist, bietet es sich an, die Abhängigkeiten integriert im Prozess zu modellieren.

Bedenkt man die bisher getroffenen Entwurfsentscheidungen, so gilt es, eine möglichst einheitliche Abhängigkeitsnotation für Fragmente und Attribute zu finden. Dazu muss man zwischen beiden Zwecken differenzieren, die Abhängigkeitsdefinitionen erfüllen. Zum einen ist dies, eine Reihenfolge der Entscheidungen bei der Variantenableitung zu bestimmen, zum

5. Variabilitätskonzept

anderen, die Alternativen eines Variabilitätspunkts aufgrund zuvor getroffener Entscheidungen einzuschränken. Bei den Variabilitätsbeschreibungen entsprechen diese zum einen der `source/target-Relation`, zum anderen den `enablingConditions`.

Die jeweils betroffenen Elemente sind bei der `source/target-Relation` variable Regionen und Attribute, bei den `enablingConditions` sind es Prozessfragmente und Alternativen in den Attributen.

Um diese zu definieren, ist es nötig, eindeutige Bezeichner für die Elemente zu vergeben, die als Variabilitätspunkte fungieren. Für die variable Region ist dies ihr Name. Für das Attribut ist es der Name des enthaltenden Elements, gefolgt vom Attributnamen nach einem Doppelkreuz. Ist beispielsweise das Attribut „Target“ eines Fragment-Links namens „Link1“ variabel, so lautet sein Bezeichner „Link1#Target“.

Für die Angabe von Abhängigkeiten wird die variable Region um folgende Attribute erweitert:

Dependencies Eine Liste der Variabilitätspunkte, von denen diese variable Region abhängt.

Fragments Eine Liste von Fragmenten, mit denen die variable Region bei der Ableitung ersetzt werden darf. Jedes dieser Fragmente kann zudem mit einer Enabling Condition versehen werden.

Damit ist es möglich, alle notwendigen Abhängigkeiten zu modellieren, ohne die Prozessfragmente zu verändern. Dies ist wichtig, da ein Prozessfragment in anderen variablen Regionen bzw. Referenzprozessen wiederverwendet werden kann.

Analog dazu muss die Attributsprache wie folgt erweitert werden, um ebenfalls Dependencies und Enabling Conditions zu erlauben:

```
Attribute ::= <Variable_Attribute>|<Fixed_Attribute>
```

```
Variable_Attribute ::= \{"alternatives":<Alternative_List>,  
  "dependencies":<Dependency_List>\}
```

```
Fixed_Attribute ::= <String>
```

```
Alternatives_List ::= [<Alternative> {, <Alternative>}*
```

```
Alternative ::= \{<Fixed_Alternative>\}|\{<Free_Alternative>\}
```

```
Fixed_Alternative ::= "type":"E","text": "<String>","ec": "<Enabling_Condition>"
```

```
Free_Alternative ::= "type":"F","text": "<Constraint>","ec": "<Enabling_Condition>"
```

```
<Constraint> ::= siehe Constraints des Variability Descriptor
```

```
<Enabling_Condition> ::= siehe Enabling Conditions des Variability Descriptor
```

```
<Dependency_List> ::= [<VarPoint_Name> {, <VarPoint_Name>}*]  
<VarPoint_Name> ::= "<String>"
```

Listing 5.2: Um Abhängigkeiten erweiterte Attributsprache, hinzugefügte Teile sind hervorgehoben

Eine Frage, die sich nun stellt, ist, ob es möglich sein soll, Abhängigkeiten zwischen Fragment und Hauptprozess zu definieren. Hier ist wieder eine Unterscheidung zu treffen.

Für die Ableitungsreihenfolge, also die Quelle-Ziel-Relation ist es nicht sinnvoll, diese Abhängigkeit zu erlauben. Da das gesamte Fragment einen Variabilitätspunkt im Referenzprozess repräsentiert, ist davon auszugehen, dass mit der Ableitung dieses Punkts (also dem Einsetzen des Fragments), dieses auch in seiner Gesamtheit abgeleitet werden muss, sodass die Ableitung des Punktes im Referenzprozess erst abgeschlossen ist, wenn die variable Region durch ein konkretes Fragment ersetzt ist. Da Fragment-Links in Prozessfragmenten variable Ziele haben können, können die Links erst aufgelöst werden, wenn die entsprechende Ableitung vorgenommen wurde. Auch dies spricht dafür, die Ableitung eines variablen Fragments unmittelbar nach seiner Einsetzung vorzunehmen.

Für Enabling Conditions wird die fragmentübergreifende Definition erlaubt, um dem Referenzprozess Kenntnis der eingesetzten Fragmente zu ermöglichen. So kann z.B. eine spätere Entscheidung im Hauptprozess davon abhängig gemacht werden, wie ein variables Fragment abgeleitet wurde oder, ob ein spezielles Element im variablen Fragment existiert. Bei diesen Definitionen ist allerdings zu beachten, dass eine solche Prüfung fehleranfällig ist, da dem Fragment a priori nicht bekannt ist, in welchen Referenzprozess es eingesetzt wird. Ebenso muss im Referenzprozess jedes mögliche einzusetzende Fragment betrachtet werden.

Um dies zu ermöglichen, wird festgelegt, dass die Elemente des Fragments und des Referenzprozesses einen Namensraum teilen. Somit können sie sich in Enabling Conditions gegenseitig referenzieren.

Bei der Definition der Enabling Conditions ist zu beachten, dass es dabei zu Deadlocks kommen kann. Eine automatisierte Überprüfung auf solche Deadlocks ist nicht möglich, da sich das Problem auf das der Erfüllbarkeit prädikatenlogischer Formeln reduzieren lässt und somit unerfüllbar ist. Gäbe es nur explizite Alternativen, wäre das Problem NP-vollständig, was heißt, dass bisher nur Lösungen existieren, die das Problem in exponentiellem Zeitaufwand lösen. Man könnte eine solche Untersuchung durch Backtracking oder ein ähnliches Verfahren durchführen. Bedenkt man aber, dass bei freien Alternativen potenziell endlos viele Möglichkeiten bestehen, ist klar, dass eine umfassende Analyse nicht möglich ist. Deswegen ist es die Pflicht des Modellierers, sicherzustellen, dass kein Deadlock auftritt. Dies kann z.B. erreicht werden, wenn an jedem Variabilitätspunkt eine Alternative ohne Enabling Condition zur Verfügung steht.

Abhängigkeitssprache

Wie bereits in 3.2 beschrieben, können in Variabilitätsbeschreibungen beliebige *XPath*-Ausdrücke verwendet werden, um Constraints und Enabling Conditions zu definieren. Für die Beschreibung im Editor ergibt sich hierbei ein Konflikt: Einerseits liegt zum Zeitpunkt der Bearbeitung noch kein XML-Dokument vor, auf das ein solcher Ausdruck angewendet werden kann, sodass der Ansatz eigentlich nicht geeignet ist. Andererseits ist ein Ziel der Modellierung der Export in eine Variabilitätsbeschreibung, wodurch sich die Verwendung von *XPath*-Ausdrücken anbietet.

Ein Kompromiss zwischen diesen beiden Anforderungen stellt eine Erweiterung von *XPath* um BPMN-spezifische Funktionen dar. Mit diesen Funktionen lassen sich BPMN-Elemente unabhängig des zugrundeliegenden Dokumentes untersuchen. So kann gewährleistet werden, dass der Ausdruck sowohl im Editor als auch beim Export funktioniert. Variabilitätsbeschreibungen haben mit `variabilityPoint` und `selectedValue` bereits zwei solche Funktionen.

Dies hat den Vorteil, dass der Nutzer die gesamte *XPath*-Syntax nutzen kann, unter anderem was Datentypen, Funktionen und logische Operatoren angeht. Für Firefox existiert bereits ein *XPath*-Parser[Gra], der von Oryx verwendet wird. Die Funktionen können dann in einem Vorbearbeitungsschritt aufgelöst werden, der sich je nach Verwendung des Ausdrucks (Auswertung oder Export) unterscheidet.

Folgende Funktionen werden zusätzlich zu den bereits vorhandenen definiert:

exists(elementName) Prüft, ob ein BPMN-Element mit dem Namen `elementName` im Diagramm enthalten ist.

elementAttribute(elementName, attributeName) Gibt das Attribut `attributeName` des Elements `elementName` als String zurück. Existiert das Attribut nicht, wird der String `undefined` zurückgegeben

5.3. Variantenableitung

Ziel der Variantenableitung ist es, aus dem Referenzprozess eine gültige Variante zu erstellen, wobei die modellierten Abhängigkeiten eingehalten werden.

Zur Variantenableitung gibt es grundsätzlich zwei Möglichkeiten: Zum einen ist dies die Ableitung über eine exportierte Variabilitätsbeschreibung. Diese kann man mit den vorhandenen Werkzeugen entweder mittels einer Art Wizard oder automatisiert ableiten.

Zum anderen soll unter Anleitung des Referenzprozessmodellierers eine Variante grafisch aus dem Referenzprozess ableitbar sein. Dies ermöglicht den Fachanwendern, augenblicklich

die Auswirkungen ihrer Entscheidung auf den Prozess zu sehen. Mittels Abhängigkeitsdefinitionen lässt sich so ein verständlicher Durchlauf durch den Referenzprozess erstellen, an dessen Ende eine fertige Variante steht.

Dabei soll ein maximaler Freiheitsgrad ermöglicht werden, d.h. der Anwender hat zu jedem Zeitpunkt die Wahl zwischen allen Variabilitätspunkten, deren Abhängigkeiten erfüllt sind. In einem korrekten Prozess ergeben die Abhängigkeiten einen gerichteten, azyklischen Graphen, der die Halbordnung zwischen den einzelnen Variabilitätspunkten beschreibt. Die Variabilitätspunkte, deren Abhängigkeiten erfüllt sind, haben darin den Eingangsgrad Null.

Dieser *Dependency-Graph* ist wie folgt definiert:

$$G = (V, E)$$

$$V = \{vp | vp \text{ ist ein Variabilitätspunkt}\}$$

$$E = \{(s, t) | (s, t) \in \text{Source-Target-Relation}\}$$

Dabei stellt die Source-Target-Relation die Abhängigkeiten dar, die vom Benutzer entweder im Attribut *Dependencies* der variablen Region oder in den variablen Attributen definiert sind. Ein Tupel (s, t) ist genau dann in der Relation, wenn in t eine Abhängigkeit zu s definiert wurde.

Analog zu einer topologischen Sortierung[CLRS04] werden nun sukzessive Knoten mit Eingangsgrad Null aus dem Graphen entfernt, wenn der Nutzer sie ableitet. So entstehen neue Wahlmöglichkeiten, da der Eingangsgrad der verbliebenen Knoten zum Teil sinkt.

Eine besondere Behandlung benötigen hierbei Prozessfragmente, die ihrerseits Variabilität enthalten. Wie oben festgestellt ist es sinnvoll, das gesamte Prozessfragment abzuleiten, sodass letzten Endes ein konkretes Prozessfragment eingesetzt wird. Um dies zu ermöglichen, wird der Ableitungsalgorithmus rekursiv auf dem variablen Prozessfragment durchgeführt, sodass die Ableitung des Hauptprozesses erst fortgesetzt werden kann, wenn das Prozessfragment vollständig abgeleitet ist.

Mit diesen Vorüberlegungen ergibt sich folgender Algorithmus.

Variante ableiten

1. Generiere eine Liste der Variabilitätspunkte, d.h. aller variablen Attribute und Regionen im abzuleitenden Prozess (Prozessfragment oder Referenzprozess)
2. Erstelle einen Graphen aller Abhängigkeiten zwischen diesen Variabilitätspunkten. Die Abhängigkeiten ergeben sich aus der Source-Target-Relation.

5. Variabilitätskonzept

3. Ist dieser Dependency-Graph zyklisch, verweigere die Ableitung und beende den Algorithmus.
4. Wiederhole, solange der Graph nicht leer ist:
 - a) Selektiere alle Variabilitätspunkte, deren Eingangsgrad im Graph Null ist. Dies sind die Punkte, für die eine Entscheidung getroffen werden kann.
 - b) Für jeden dieser Variabilitätspunkte:
 - i. Falls nur eine Wahl möglich ist, triff diese Wahl automatisch, entferne den Punkt aus dem Graph, brich die Iteration ab und überspringe Schritt c und d.
 - ii. Gibt es mehr als eine Wahlmöglichkeit, mache den Variabilitätspunkt für den Nutzer wählbar.
 - c) Lass den Nutzer eine Wahl für einen der wählbaren Punkte treffen (siehe Unterprozess)
 - d) Entferne den gewählten Punkt aus dem Graphen.
5. Die Variante ist fertig abgeleitet. Ermögliche Speicherung, Deployment, etc.

Unterprozess Wahl treffen

1. Zeige eine Liste aller Alternativen (Attributwerte oder Prozessfragmente) für den gewählten Variabilitätspunkt, deren Enabling Condition erfüllt ist.
2. Lass den Nutzer eine davon wählen.
3. Wurde eine freie Alternative gewählt, überprüfe den Constraint. Ist er nicht erfüllt, beginne den Unterprozess von vorne.
4. Setze die getroffene Entscheidung um. Im Falle eines Attributes trage den entsprechenden Wert ein. Im Falle eines Prozessfragments setze es in den Referenzprozess ein.
5. Wurde ein Prozessfragment gewählt, prüfe, ob es Variabilität enthält.
 - Falls ja, rufe Variante ableiten für dieses Prozessfragment auf.
6. Wurde ein Prozessfragment gewählt und enthält es Fragment-Links, löse diese auf.

Dieser Algorithmus beschreibt das Vorgehen bei der Ableitung. Ungeklärt sind allerdings die Behandlung von Fehlern (fehlerhafte Fragmente, nichtvorhandene Links, etc.) und die zu verwendenden Datenstrukturen. Dieser Feinentwurf wird in der Implementierung vorgenommen.

6. Implementierung

Zur Umsetzung des im letzten Kapitels vorgestellten Variabilitätskonzepts wird der Editor Oryx verwendet, dessen Erweiterungsmöglichkeiten bereits in 3.3 beschrieben wurden.

Analog zur Beschreibung des Variantenkonzepts wird die Implementierung ebenfalls in Variabilitätsmodellierung, Abhängigkeitsmodellierung und Variantenableitung unterteilt.

Weitere Bilder der fertigen Erweiterung und eine Benutzungsanleitung befinden sich im Anhang A.3.

6.1. Übersicht

Abbildung 6.1 zeigt ein FMC-Diagramm[WWWq], das die vorgenommenen Erweiterungen in die Oryx-Architektur einordnet. Es ist angelehnt an das FMC-Diagramm der Oryx-Architektur, die sich in [Tsc07] in Abschnitt 4.2 findet.

Der folgende Abschnitt zählt kurz die vorzunehmenden Erweiterungen auf, die im Folgenden näher beschrieben werden.

Stencil Set Extensions

Die beiden implementierten Stencil Set Extensions dienen dazu, im Editor die nötigen neuen Diagrammkomponenten zur Verfügung zu stellen.

Stencil Set Variability Erweitert BPMN 2.0 um die variable Region.

Stencil Set Fragments Erweitert BPMN 2.0 und das *Stencil Set Variability* zusätzlich um Fragment-Start, -Link und -Ende.

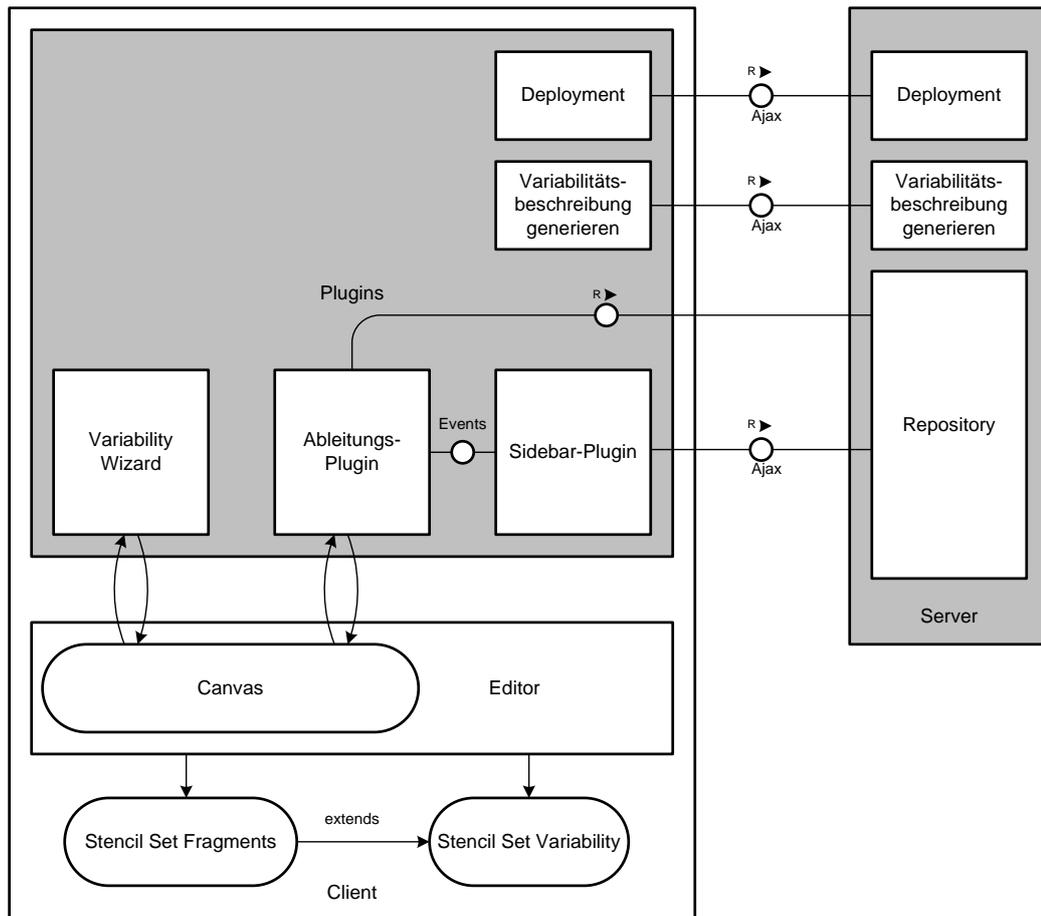


Abbildung 6.1.: Architektur der Oryx-Erweiterung. (vergleiche Architektur Oryx[Tsco7])

Client-Plugins

Variability Wizard Erlaubt die einfache Erstellung von variablen Attributen, Abhängigkeiten und Enabling Conditions.

Ableitungs-Plugin Ermöglicht die Ableitung einer Variante im Editor.

Sidebar-Plugin Ermöglicht die Darstellung von Alternativen in einer Seitenleiste des Editors. Wird im Rahmen der Ableitung vom *Ableitungs-Plugin* verwendet.

Variabilitätsbeschreibung generieren Ermöglicht den Export des Referenzprozesses als Diagramm mit Variabilitätsbeschreibung.

Deployment Überprüft, ob eine Variante bereit für das Deployment ist und schickt sie an den Server.

Server-Plugins

Variabilitätsbeschreibung generieren Serverseitige Komponente des Exports.

Deployment Platzhalter für das Deployment in eine Process Engine.

6.2. Variabilitätsmodellierung

Bei der Variabilitätsmodellierung sind zwei Aspekte zu unterscheiden. Zum einen gilt es, Modellierung von Diagrammen mit der BPMN-Erweiterung zu ermöglichen. Zum anderen müssen variable Attribute behandelt werden.

6.2.1. Umsetzung der BPMN-Erweiterung

Zur Umsetzung der beschriebenen BPMN-Erweiterung für Referenzprozesse und Prozessfragmente bietet sich die Verwendung von Stencil Set Extensions an. Diese können auf dem bereits existierenden BPMN 2.0-Stencil Set aufbauen.

Für den Referenzprozess muss lediglich das Element *Variable Region* hinzugefügt werden. Die Regeln für die Einbindung in den Sequenzfluss (z.B. genau ein Ein- und Ausgang) lassen sich mittels Oryx modellieren.

Für Prozessfragmente müssen darüber hinaus noch Fragment-Start, Fragment-Ende und Fragment-Link zu Verfügung gestellt werden. Außerdem müssen einige Elemente des Original-Stencil Sets entfernt werden (z.B. Pools und Lanes). Auch dies lässt sich mit Oryx umsetzen.

Mittels dieser beiden Stencil Set Extensions kann die beschriebene Erweiterung modelliert werden.

6.2.2. Umsetzung variabler Attribute

Für die Beschreibung variabler Attribute ist wie oben erwähnt ein Wizard wünschenswert, der es dem Nutzer erspart, die Ausdrücke selbst zu schreiben. Dieser kann in Oryx als Client-Plugin umgesetzt werden.

Dieser *Variability Wizard* erscheint als Werkzeug in der Oryx-Toolbar. Wird es aktiviert, zeigt es die Attribute des gewählten Elements und ermöglicht es, jedes davon zu einem variablen Attribut zu machen. Ein Wizard erlaubt hierbei das Erstellen einzelner Alternativen und generiert automatisch einen Ausdruck in der Variabilitätssprache (siehe 5.2). Abbildung 6.2 zeigt ein Beispiel für ein variables Attribut im *Variability Wizard* und den generierten Ausdruck.

6. Implementierung

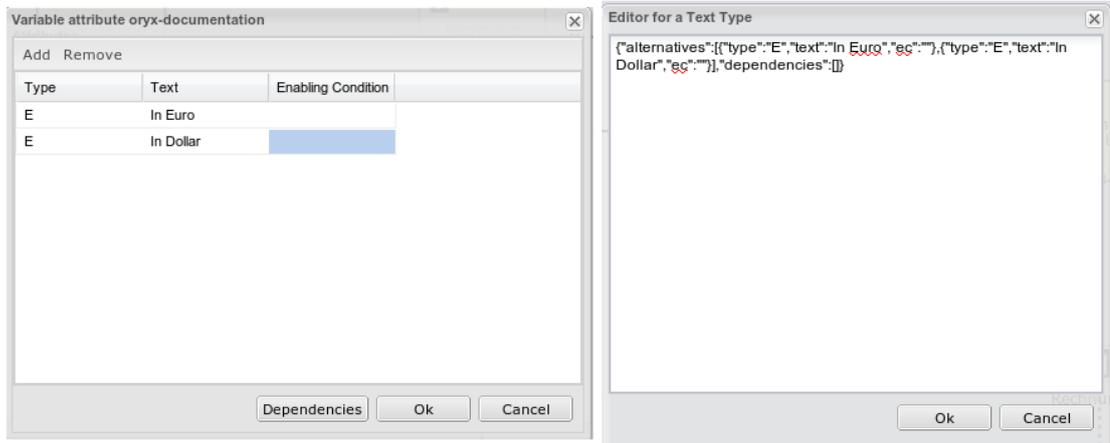


Abbildung 6.2.: Erstellung eines variablen Attributs. Links im *Variability Wizard*, zum Vergleich rechts der generierte Ausdruck.

Zusätzlich zur Definition von variablen Attributen erlaubt der *Variability Wizard* eine komfortablere Definition der Attribute einer variablen Region. Zwar lassen sich die Liste der wählbaren Fragmente und die Abhängigkeiten in Oryx auch als Listen-Attribut der variablen Region bearbeiten, aber dazu ist eine präzise Eingabe von Fragment- und Attributnamen notwendig. Der Wizard gibt dem Nutzer eine Checkliste aller vorhandenen Fragmente, in denen er die markieren kann, die in diese variable Region eingesetzt werden dürfen. Für jedes von ihnen lässt sich eine Enabling Condition definieren.

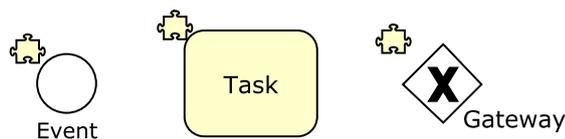


Abbildung 6.3.: Darstellung von Elementen mit variablen Attributen.

Elemente mit variablen Attributen werden wie in Abbildung 6.3 dargestellt mit einem Piktogramm versehen, um es dem Nutzer zu erleichtern, sie zu erkennen. Dies stellt keine Erweiterung des Stencil Sets dar, sondern wird innerhalb des Plugins mittels sogenannter Overlays umgesetzt.

6.3. Abhängigkeitsmodellierung

Die Modellierung von Abhängigkeiten erfolgt ebenfalls mithilfe des *Variability Wizard*.

Da ein variables Attribut einen Variabilitätspunkt darstellt, können Abhängigkeiten zu anderen Variabilitätspunkten festgelegt werden. Dazu bietet das Plugin eine Checkliste aller Variabilitätspunkte im Diagramm an. Wie das variable Attribut stellt auch die variable Region einen Variabilitätspunkt dar. Der Wizard ermöglicht auch hier die Wahl von Abhängigkeiten.

Enabling Conditions können im Falle einer variablen Region bei der Wahl der Fragmente oder im Falle eines variablen Attributs beim Erstellen der Alternative eingegeben werden. Wegen der Ausdrucksmächtigkeit von XPATH muss die Enabling Condition von Hand geschrieben werden. Ein grafischer Erstellungsprozess ist für eine zukünftige Ausbaustufe des Plugins denkbar.

6.4. Variantenableitung

In Abschnitt 5.3 wurde bereits der Algorithmus zur Variantenableitung beschrieben. Die Beschreibung der Implementierung orientiert sich am Ablauf des Algorithmus. Die Umsetzung dieses Prozesses wird auf zwei Komponenten aufgeteilt.

Das *Ableitungs-Plugin* ist für die eigentliche Ableitung zuständig und enthält die notwendigen Algorithmen und Datenstrukturen. Für die Wahl der Alternativen wird eine Leiste am Rand des Editors eingesetzt. Diese Leiste ist im *Sidebar-Plugin* realisiert. Plugins in Oryx kommunizieren über ein Event-System, mittels dem Komponenten durch Publish/Subscribe[GHJV95] kommunizieren können.

6.4.1. Dependency-Graph

Wie im Konzept beschrieben (siehe 5.3) ist der Dependency-Graph ein gerichteter azyklischer Graph, dessen Knoten die Variabilitätspunkte und dessen Kanten deren Abhängigkeiten zueinander sind. Aktiviert der Nutzer das Ableitungsplugin, wird er aus dem geladenen Diagramm konstruiert.

Abbildung 6.4 zeigt beispielhaft, wie der Nutzer die Dependencies festlegen kann. In einem Diagramm mit drei variablen Regionen VP_1 , VP_2 und VP_3 wählt er im *Variability Wizard*, dass VP_2 von den anderen beiden variablen Regionen abhängt. Das Resultat ist der abgebildete Dependency-Graph.

Um die Anforderungen an die Datenstruktur des Dependency-Graphs zu ermitteln, wird betrachtet, welche Operationen darauf im beschriebenen Algorithmus durchgeführt werden:

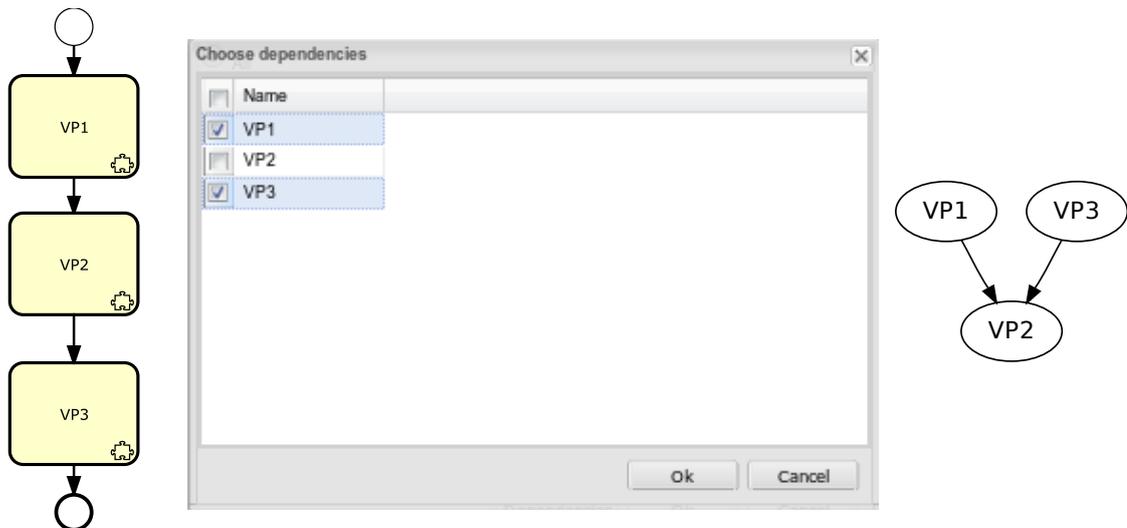


Abbildung 6.4.: Definition des Dependency-Graph durch den Benutzer. Links das Diagramm, in der Mitte die definierten Dependencies im *Variability Wizard*, rechts der resultierende Dependency-Graph.

Zugriff Die Datenstruktur muss es ermöglichen, anhand eines Elements auf dessen zugehörigen Variabilitätspunkte zuzugreifen. Dies ist notwendig, um die Variabilitätspunkte aus der Nutzerwahl zu ermitteln.

Entfernen Ein Variabilitätspunkt muss mit allen zugehörigen Kanten aus dem Graph entfernt werden. Dies geschieht jedes Mal, wenn eine gültige Ableitungsentscheidung getroffen wurde.

Auswahl Auswahl aller Variabilitätspunkte mit Eingangsgrad Null.

Mit diesen Vorüberlegungen fällt die Entscheidung für die Datenstruktur auf eine nach Elementen sortierte Adjazenzliste, die sich wie folgt zusammensetzt:

DependencyGraph Der `DependencyGraph` enthält eine Hash-Liste aus `DepElements`, die jeweils ein Element des Diagramms repräsentieren.

DepElement Jedes `DepElement` enthält wiederum eine Anzahl von `DepVarPoints`, die die einzelnen Variabilitätspunkte und damit die Knoten des Graphs repräsentieren.

DepVarPoint Jeder `DepVarPoint` enthält eine Liste von anderen `DepVarPoints`, die zu ihm abhängig sind. Im Graph sind diese die Ziele seiner ausgehenden Kanten. Zusätzlich wird in jedem `DepVarPoint` sein Eingangsgrad mitgeführt, also die Anzahl der anderen Punkte, die auf ihn verweisen.

Mit dieser Datenstruktur lassen sich alle drei Operationen effizient umsetzen. Die Hash-Liste stellt den schnellen *Zugriff* auf die Variabilitätspunkte eines Elements sicher. Das *Entfernen* eines Variabilitätspunkts erfolgt, indem der entsprechende `DepVarPoint` entfernt wird, nachdem allen ausgehenden Kanten gefolgt wurde und bei den Ziel-Punkten der Eingangsgrad um Eins gesenkt wurde. Die *Auswahl* aller Variabilitätspunkte mit Eingangsgrad Null hingegen benötigt nur einen Durchlauf über alle `DepVarPoints`, da jeweils nur der gespeicherte Eingangsgrad geprüft werden muss.

Für die Umsetzung des rekursiven Teils des Algorithmus, in dem ein variables Fragment abgeleitet wird, sind weitere Operationen vonnöten. Der rekursive Aufruf lässt sich nicht ohne Weiteres in Oryx abbilden: Zum einen steht nur eine Leinwand zur Verfügung, sodass das abzuleitende Fragment nicht separat vom Referenzprozess angezeigt werden kann. Dies wäre nur über die Verwendung mehrerer Oryx-Sessions möglich, die allerdings nicht miteinander kommunizieren können. Zum anderen entspricht der Kontrollfluss des Ableitungsalgorithmus nicht einem Methodendurchlauf. Durch die Interaktion mit dem Nutzer setzt er sich aus verschiedenen Methodenaufrufen des Plugins zusammen. Deswegen müsste die Rekursion manuell umgesetzt werden, indem man beispielsweise einen Stack aus Dependency-Graphen erzeugt und diese dann abarbeitet.

Eine einfachere Alternative ist es, beim Importieren eines Fragments die Dependency-Graphen des Fragments und des Referenzprozesses zu vereinigen. Dabei müssen zusätzliche Kanten eingefügt werden, um sicherzustellen, dass zuerst die Variabilitätspunkte des Fragments abgeleitet werden und erst danach die Ableitung des Referenzprozesses fortgesetzt werden kann. Der naive Ansatz, jeden Knoten des Dependency-Graphen des Fragments mit jedem des Graphen des Referenzprozesses zu verbinden, führt zu $n * m$ zusätzlichen Kanten, wobei n und m die Anzahl der Variabilitätspunkte des Fragments bzw. des Referenzprozesses sind. Mit mehrfachem rekursiven Einsetzen steigt die Zahl dieser Kanten weiter an.

Um diese erhöhte Kantenanzahl und den damit verbundenen Rechenaufwand zu vermeiden, wird ein zusätzlicher Variabilitätspunkt eingefügt, der im Folgenden als Pivot-Punkt bezeichnet wird. Dieser Pivot-Punkt dient als Separator zwischen dem Dependency-Graph des importierten Fragments und dem des Referenz-Prozesses. Um sicherzustellen, dass zuerst alle Variabilitätspunkte des Fragments abgeleitet werden, bevor die Ableitung des Referenz-Prozesses fortgesetzt werden kann, wird der Pivot-Punkt wie folgt in den Graphen eingebunden: Jeder einzelne Variabilitätspunkt des Fragments ist eine Vorbedingung für die Ableitung des Pivot-Punkts. Der Pivot-Punkt ist seinerseits eine Vorbedingung für die Ableitung aller Variabilitätspunkte im Referenzprozess. Durch die Transitivität der Ableitungsreihenfolge ist somit sichergestellt, dass das Fragment vollständig abgeleitet wird, bevor die Ableitung des Referenzprozesses fortgesetzt wird. Im Gegensatz zum naiven Ansatz müssen hier nur $n + m$ zusätzliche Kanten hinzugefügt werden.

Abbildung 6.5 zeigt einen Vergleich der beiden Methoden. In den Referenzprozess mit drei Variabilitätspunkten R_1 , R_2 und R_3 wird ein Fragment mit den Variabilitätspunkten F_1 , F_2 und F_3 eingesetzt. Mit Verwendung des Pivot-Punkts sind für die zusätzlichen

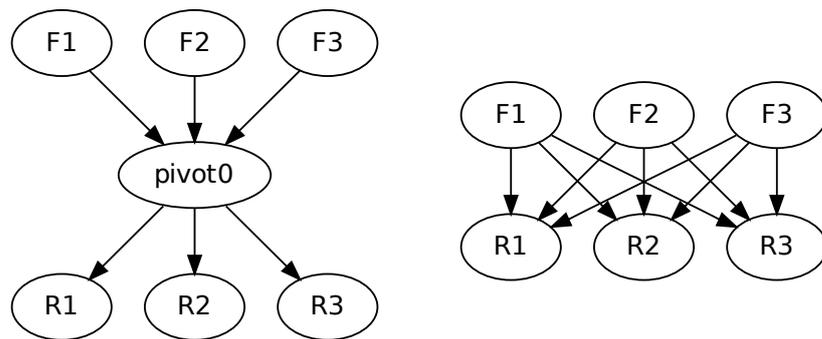


Abbildung 6.5.: Dependency-Graph mit (links) oder ohne Verwendung (rechts) eines Pivot-Punkts.

Abhängigkeiten nur $3 + 3 = 6$ Kanten notwendig, wohingegen es beim naiven Ansatz $3 * 3 = 9$ Kanten sind.

Folgendes Beispiel illustriert die Verwendung des Pivot-Punkts:

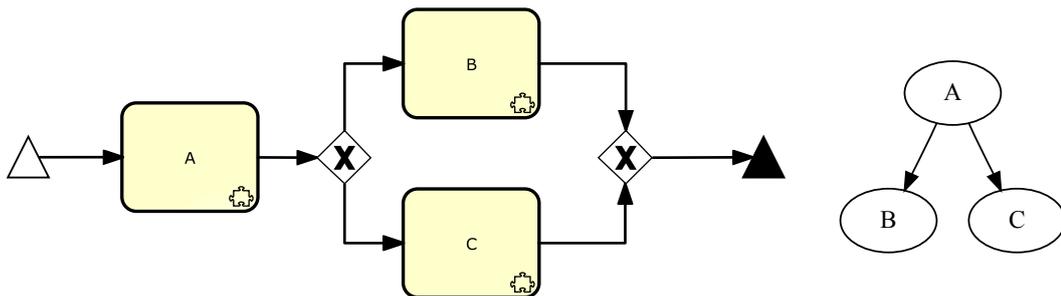


Abbildung 6.6.: Prozessfragment mit zugehörigem Dependency-Graph.

Abbildung 6.6 zeigt ein Prozessfragment mit zugehörigem Dependency-Graph. Das Fragment enthält drei variable Regionen und damit drei Variabilitätspunkte. Der Dependency-Graph besagt, dass Region *A* abgeleitet werden muss, bevor die Regionen *B* und *C* abgeleitet werden können.

In Abbildung 6.7 ist ein Referenzprozess zu sehen, der vier variable Regionen enthält. Die Regionen *S*, *X* und *Y* müssen abgeleitet werden, bevor die Region *Z* abgeleitet werden kann.

Bei der Ableitung wird nun die variable Region *S* im Referenzprozess (Abbildung 6.7) mit dem Prozessfragment (Abbildung 6.6) abgeleitet. Abbildung 6.8 zeigt das Diagramm und den Dependency-Graph nach der Einsetzung. Der Variabilitätspunkt von *S* wurde aus dem

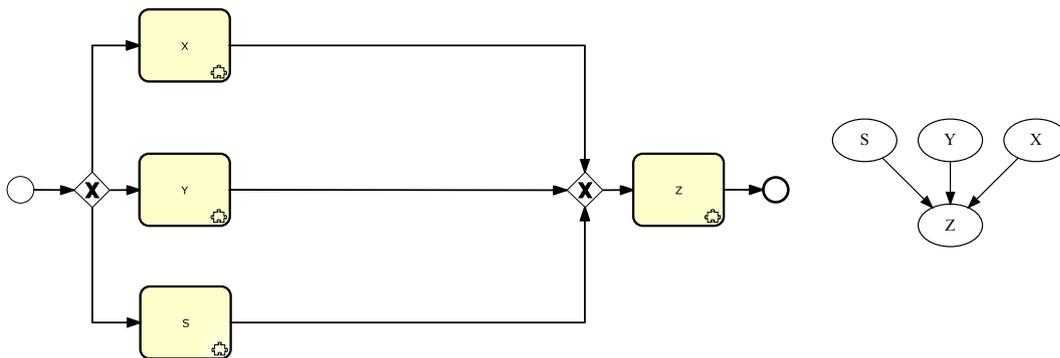


Abbildung 6.7.: Referenzprozess mit zugehörigem Dependency-Graph.

Graph entfernt, da die Ableitung vorgenommen wurde. Der Dependency-Graph des Fragments wurde dem Graphen des Referenz-Prozesses hinzugefügt. Da die Variabilitätspunkte des Fragments zuerst abgeleitet werden müssen, wurde dem Graph ein Pivot-Punkt mit Namen *pivot0* hinzugefügt. Der Pivot-Punkt trennt die beiden zusammengesetzten Graphen voneinander. Er kann erst nach A, B, C abgeleitet werden. X, Y, Z können erst abgeleitet werden, nachdem der Pivot-Punkt abgeleitet wurde. Damit ist sichergestellt, dass das Fragment vollständig abgeleitet wird, bevor die Ableitung des Hauptprozesses fortgesetzt werden kann.

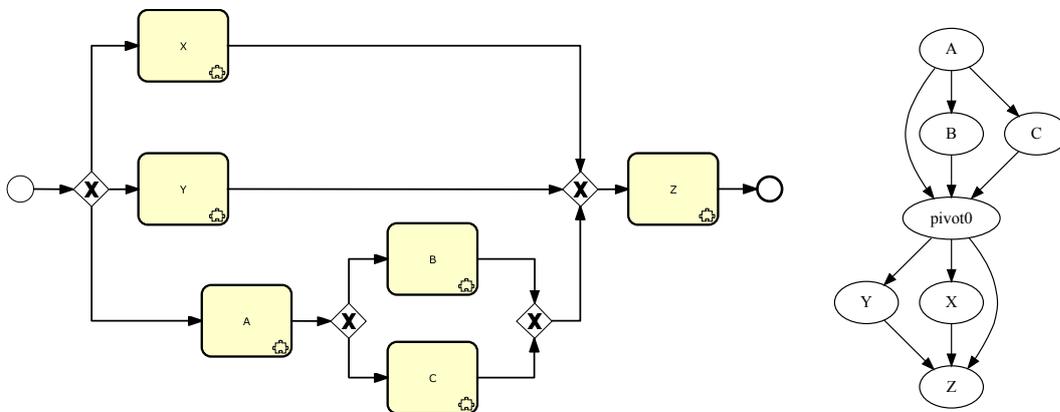


Abbildung 6.8.: Referenzprozess und Dependency-Graph nach Einsetzung des Prozessfragments.

6.4.2. Ableitungsmodus

Aktiviert der Nutzer das *Ableitungs-Plugin*, so schaltet Oryx in den sog. Ableitungsmodus. In diesem Modus verhindert das Plugin die weitere Überarbeitung des Diagramms und führt den Ableitungsalgorithmus aus. Zuerst wird wie oben beschrieben der Dependency-Graph erstellt, indem das Diagramm auf Variabilität durchsucht wird.

Aufbauend auf dem Graphen markiert das Plugin die Elemente, die Variabilität erhalten. Elemente, die mindestens einen ableitbaren Variabilitätspunkt enthalten, werden grün markiert. Elemente, die zwar Variabilitätspunkte enthalten, von denen aber keiner ableitbar ist, markiert das Plugin rot. Diese Markierungen werden mit jeder getroffenen Entscheidung anhand des aktuellen Dependency-Graphen aktualisiert. Sind alle Variabilitätspunkte eines Elements abgeleitet, wird die Markierung entfernt.

Der Nutzer kann nun mit einem Mausklick eines der Elemente wählen, das ableitbare Variabilitätspunkte enthält. Sobald er dies tut, wird ein Event an das *Sidebar-Plugin* geschickt, das die weitere Handhabung übernimmt.

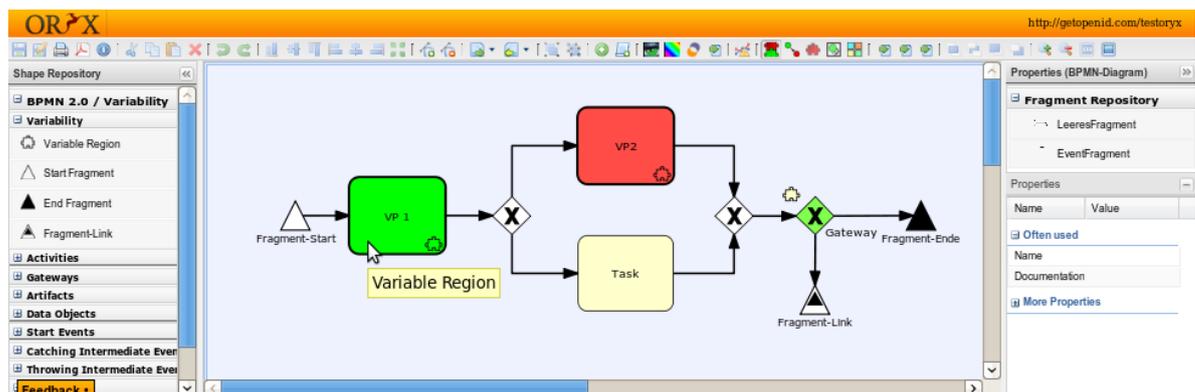


Abbildung 6.9.: Oryx im Ableitungsmodus.

Abbildung 6.9 zeigt den Editor im Ableitungsmodus. Die variable Region *VP1* und ein variables Attribut im rechten Gateway können bereits abgeleitet werden und sind grün markiert. Die Dependencies für die variable Region *VP2* sind noch nicht erfüllt, weswegen sie rot hinterlegt ist. Auf der rechten Seite befindet sich die *Sidebar*, die die beiden Alternativen für *VP1* anzeigt.

6.4.3. Auswertung von Constraints und Enabling Conditions

Sowohl bei der Auflistung von Entscheidungen in der Sidebar als auch bei der Überprüfung auf automatische Ableitbarkeit müssen Constraints und Enabling Conditions in Bezug auf den aktuellen Zustand des Diagramms und des Ableitungsprozesses ausgewertet werden.

Diese Auswertung erfolgt in drei Schritten:

1. Handelt es sich um einen Constraint, wird die Variable `$value` durch den vom Nutzer gewählten Wert ersetzt.
2. Danach werden die in 5.2 beschriebenen Funktionen ausgewertet und im String durch ihre Ergebnisse ersetzt.
3. Der vorverarbeitete Ausdruck wird dann an einen Firefox-internen XPath-Parser[Gra] übergeben, der ihn auswertet.

Um die Funktionen auszuwerten, die eventuell auch ineinander geschachtelt sein können, wird ein generischer `PreParser` verwendet. Dieser `PreParser` erhält einen sog. `RuleContext`, in dem die Namen der zu suchenden Funktionen und ihre Implementierung enthalten sind. Dieser generische Ansatz hat den Vorteil, dass die Parser-Logik mit anderen Funktionen und anderen Implementierungen der Funktionen wiederverwendet werden kann, z.B. beim Export des Diagramms in eine Variabilitätsbeschreibung. Außerdem können so neue Funktionen hinzugefügt werden, indem der Kontext erweitert wird. Dies entspricht dem sogenannten Strategy-Pattern[GHJV95].

Für das Parsen erstellt der `PreParser` aus den aufzulösenden Funktionen einen regulären Ausdruck. Damit durchsucht er den String nach Vorkommnissen der Funktionen und ruft sich für ihre Parameter rekursiv selbst auf. So entsteht ein Baum, mittels dessen die Funktionsaufrufe in der Klammerungsreihenfolge durchgeführt werden: die innersten zuerst, die äußersten zuletzt.

Kann ein Ausdruck nicht ausgewertet werden, wird dem Nutzer ein Fehler ausgegeben.

Anhand des folgenden Ausdrucks werden die drei Verarbeitungsschritte beispielhaft erläutert.

```
$value != selectedValue(variabilityPoint(Task1#oryx-documentation))
```

Dieser Constraint besagt, dass der vom Nutzer eingegebene Wert ungleich des gewählten Werts im Variabilitätspunkt `Task1#oryx-documentation` sein muss. Dieser Wert sei in unserem Beispiel „17“.

Nun gibt der Nutzer bei der Ableitung den Wert „16“ ein. Bevor die Eingabe akzeptiert werden kann, wird der Constraint ausgewertet. Dazu wird zuerst `$value` durch die Eingabe ersetzt, woraus folgender Ausdruck resultiert:

```
16 != selectedValue(variabilityPoint(Task1#oryx-documentation))
```

6. Implementierung

Im zweiten Schritt werden die Funktionen ausgewertet und zwar gemäß der Klammerung von innen nach außen. Daraus ergibt sich bei der Auflösung von `variabilityPoint` zuerst folgender Ausdruck:

```
16 != selectedValue(Task1#oryx-documentation)
```

Danach wird die `selectedValue`-Funktion ausgewertet:

```
16 != 17
```

Im dritten Schritt wird der vorbereitete Ausdruck an den XPATH-Parser übergeben, der als Ergebnis `TRUE` zurückliefert.

6.4.4. Sidebar

Das *Sidebar-Plugin* nimmt gegenüber des Haupt-*Ableitungs-Plugins* eine untergeordnete Rolle ein. Es dient der Verwaltung einer Leiste an der rechten Seite des Editors, in der verschiedene Daten angezeigt werden können. Die Implementierung verwendet das in Oryx vorhandene `FragmentRepository-Plugin` als Grundlage.

Wählt der Nutzer im Ableitungsmodus ein ableitbares Element, so wird die Sidebar mittels eines Events benachrichtigt. Je nachdem, um was für ein Element es sich handelt, werden nun verschiedene Wahlmöglichkeiten angezeigt.

- Enthält das Element mehr als einen ableitbaren Variabilitätspunkt (beispielsweise zwei variable Attribute), so muss zuerst einer davon gewählt werden.
- Bei einem variablen Attribut stehen die verschiedenen Alternativen zur Auswahl, deren Enabling Condition erfüllt ist. Wird eine freie Alternative gewählt, so muss der Nutzer einen Freitext eingeben, der auf den Constraint der Alternative geprüft wird.
- Bei einer variablen Region wird eine Liste der wählbaren Fragmente angezeigt, deren Enabling Condition erfüllt ist. Dafür werden aus dem Oryx-Repository entsprechende Metadaten angefordert, sodass Namen und Thumbnails der Fragmente angezeigt werden können.

Abbildung 6.10 zeigt die Sidebar mit verschiedenen Inhalten.

Trifft der Nutzer eine gültige Wahl, so wird diese von der Sidebar per Event an das *Ableitungs-Plugin* zurückgeschickt.

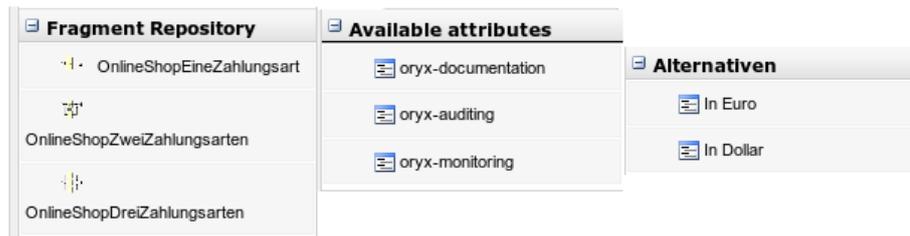


Abbildung 6.10.: Sidebar mit verschiedenen Wahlmöglichkeiten. Von links nach rechts: Fragmente, variable Attribute und Alternativen.

6.4.5. Fragment-Import

Hat der Nutzer mithilfe des *Sidebar-Plugins* eine Entscheidung getroffen, wird diese vom *Ableitungs-Plugin* umgesetzt. Handelt es sich um ein variables Attribut, so wird der gewählte Wert in das Attribut geschrieben. Wurde ein Fragment für eine variable Region ausgewählt, so muss die Region durch das Fragment ersetzt werden.

Dieser Import hat zwei Phasen. Zuerst wird das serialisierte Fragment für den Import überprüft und bearbeitet. Dann wird es in das Diagramm importiert und eingebunden. Diese Unterteilung hat den Zweck, dass bei einem Fehler vor dem Import abgebrochen werden kann, bevor das Diagramm verändert wurde.

Im Einzelnen verläuft der Import in folgenden Schritten:

1. Das zu importierende Fragment wird im JSON-Format aus dem Repository geladen.
2. Anpassung der JSON-Datei für den Import in das Diagramm.
 - a) Anpassung der X/Y-Koordinaten des Fragments an die Position der variablen Region.
 - b) Entfernen von Fragment-Start und Ende mit dem dazugehörigen Sequenzfluss.
 - c) Die an Fragment-Start und -Ende angrenzenden Elemente des Fragments werden dabei gespeichert, da dies später die Stellen sind, an denen der Referenzprozess anschließt.
3. Import des modifizierten Fragments in das Diagramm.
4. Verschiebung des importierten Fragments auf die Hierarchieebene der variablen Region. War beispielsweise die variable Region in einem Pool enthalten, so muss das Fragment ebenfalls in diesem enthalten sein.
5. Umhängen des ein- und ausgehenden Sequenzflusses der variablen Region auf die entsprechenden Elemente des importierten Fragments.

6. Ermittlung aller Fragment-Links im importierten Fragment.
7. Aktualisierung des Dependency-Graph (siehe 5.3).
8. Löschen der variablen Region aus dem Diagramm.

Ist dieser Importvorgang erfolgreich abgeschlossen, so ist das Fragment anstelle der variablen Region in das Diagramm eingebunden. Einzig die Fragment-Links sind noch nicht aufgelöst.

6.4.6. Auflösung der Fragment-Links

Die Auflösung der Fragment-Links kann nicht unmittelbar nach dem Import des Fragments geschehen, da die Links eventuell noch Variabilität enthalten. So kann beispielsweise das Ziel eines Links abhängig von vorherigen Entscheidungen bei der Ableitung des Fragments sein. Daraus ergibt sich, dass im Allgemeinen die Links erst dann aufgelöst werden können, wenn das Fragment nicht nur importiert, sondern auch vollständig abgeleitet ist.

Dieser Zeitpunkt ist genau dann gekommen, wenn der dem Fragment zugehörige Pivot-Punkt im Dependency-Graph den Eingangsgrad null hat. Daher bietet es sich an, die Fragment-Links bei der Ableitung des Pivot-Punkts aufzulösen. Dazu wird der Link entfernt und der zugehörige Fluss zum Ziel des Links umgebogen. Dies geschieht bei der automatischen Ableitung.

6.4.7. Automatische Ableitung

Nachdem das *Ableitungs-Plugin* eine vom Nutzer getroffene Entscheidung umgesetzt hat, überprüft es, ob es nun Variabilitätspunkte gibt, die automatisch abgeleitet werden können. Dazu werden die Alternativen für jeden ableitbaren Variabilitätspunkt geprüft, indem die Enabling Condition für jede Alternative ausgewertet wird. Findet sich so exakt eine Alternative mit erfüllter Enabling Condition, kann der Variabilitätspunkt automatisch abgeleitet werden. Eine Ausnahme bilden freie Alternativen. Diese können grundsätzlich nicht automatisch abgeleitet werden, da eine Nutzereingabe erforderlich ist.

Wurde ein automatisch ableitbarer Variabilitätspunkt gefunden, wird die Prüfung auf ableitbare Punkte abgebrochen. Nach der automatischen Ableitung wird die Prüfung noch einmal durchgeführt, da sich durch das veränderte Diagramm andere Wahrheitswerte für die Enabling Conditions ergeben können.

6.4.8. Protokollierung

Während der Ableitung protokolliert das *Ableitungs-Plugin* die vorgenommenen Ableitungsschritte in einem Log. Dieses Log kann über ein Icon jederzeit vom Nutzer angezeigt werden.

Außerdem wird protokolliert, welche Teile des Diagramms welchem Fragment entstammen. Dazu wird eine Liste der Elemente jedes importierten Fragments gespeichert. Nach vollendeter Ableitung kann das Diagramm in einen Anzeigemodus versetzt werden, in dem genau wie im Ableitungsmodus keine Veränderungen möglich sind. Die Elemente der einzelnen Fragmente werden in dieser Ansicht mit verschiedenen Farben markiert. In der Sidebar wird eine Legende angezeigt, sodass der Nutzer erkennen kann, welche Elemente welchem Fragment entstammen. Wird der Anzeigemodus beendet, werden die Markierungen wieder entfernt.

6.5. Export

Beim Export sind zwei Fälle zu unterscheiden. Der eine ist der Export der abgeleiteten Variante, die keine Variabilität mehr enthält. Dieser Export kann über die Oryx-eigenen Funktionen erfolgen.

Der zweite Fall ist der Export eines Referenzprozesses in eine XML-Datei im BPMN 2.0 DI-Format und eine zugehörige Variabilitätsbeschreibung. Dafür dienen wie aus dem Architekturdiagramm (siehe Abbildung 6.1) ersichtlich die beiden Plugins *Variabilitätsbeschreibung generieren*. Der Export verläuft dabei in drei Schritten:

1. Das clientseitige Plugin sendet eine Anfrage an das serverseitige Plugin, die den mit JSON serialisierten Referenzprozess enthält.
2. Auf dem Server werden die BPMN 2.0 DI-Datei und eine zugehörige Variabilitätsbeschreibung erstellt und an den Client gesendet.
3. Der Client zeigt die erhaltenen Dateien in einem separaten Fenster oder bietet sie zum Download an.

Während der Client hier nur Hilfsfunktionen übernimmt, führt das Server-Plugin den eigentlichen Exportvorgang durch.

Oryx verfügt bereits einen funktionsfähigen BPMN-Export, den man hier verwenden kann. Zusätzlich muss die Variabilitätsbeschreibung erstellt werden. Da die Oryx-eigenen Exportfunktionen nicht mit den vorgenommenen BPMN-Erweiterungen (Variable Region, etc.) umgehen können, muss die vom Client übermittelte JSON-Datei vorher modifiziert werden, sodass sie keine Variabilitätskonstrukte mehr enthält.

Da die Datei zur Erstellung der Variabilitätsbeschreibung ohnehin auf Variabilität untersucht werden muss, können somit zwei Aufgaben auf einmal erledigt werden. Die Variabilität wird also sozusagen „abgeerntet“, d.h. in dem Moment, in dem sie der Variabilitätsbeschreibung hinzugefügt wird, wird sie aus der JSON-Datei entfernt.

Für die Serialisierung der Variabilitätsbeschreibung wird JAXB (*Java Architecture for XML Binding*)[WWWg] verwendet, das in Oryx zur BPMN 2.0-Serialisierung dient. In Verbindung mit dem XJC-Compiler ist es so möglich, aus der XML-Schema-Datei für Variabilitätsbeschreibungen automatisch Java-Klassen zu erstellen. Mittels dieser kann dann die Variabilitätsbeschreibung in Java aufgebaut und automatisch in XML umgewandelt werden.

Auf der Serverseite verläuft der Export also wie folgt:

1. Die erhaltene JSON-Datei wird in ein Diagramm deserialisiert.
2. Zur Erstellung der Variabilitätsbeschreibung wird das Diagramm nach variablen Attributen und Regionen durchsucht. Beim Auffinden solch eines Variabilitätspunkts wird er im Diagramm ersetzt und der Variabilitätsbeschreibung hinzugefügt.
3. Das Resultat dieses Vorgangs sind die fertige Variabilitätsbeschreibung und ein Diagramm ohne Variabilität.
4. Sowohl das Diagramm als auch die Variabilitätsbeschreibung werden in XML serialisiert.

Die zu erstellende Variabilitätsbeschreibung soll letzten Endes die selbe Funktionalität wie die Variantenableitung bieten(siehe 6.4). Der Ableitungsalgorithmus wird dazu mit statischen Mitteln beschrieben. Ein statisches Dokument kann allerdings die Rekursion beim Ableiten von Fragmenten nur begrenzt abbilden. Es gibt in Variabilitätsbeschreibungen zwar die Möglichkeit des Zugriffs auf andere Dokumente (z.B. exportierte Fragmente), allerdings lassen sich während der Ableitung einer Beschreibung keine weiteren Variabilitätspunkte definieren bzw. nachladen. Das hat zur Folge, dass jede einzelne mögliche Entscheidung in der Variabilitätsbeschreibung enthalten sein muss. Beim rekursiven Einsetzen von Fragmenten ineinander kann deswegen die Zahl der Variabilitätspunkte exponentiell anwachsen. Ein Fragment, das in sich selbst oder eines seiner Kinder einsetzbar ist, kann folglich nicht mit endlich vielen Variabilitätspunkten abgebildet werden, da diese zyklische Einsetzung im Prinzip beliebig oft wiederholt werden kann und für jede Wiederholung zusätzliche Variabilitätspunkte in der Beschreibung notwendig sind.

Aus diesem Grund wird der Export von variablen Fragmenten nicht unterstützt. Das heißt, dass jedes Fragment, das in den Referenzprozess eingesetzt werden kann, statisch sein muss, also keine Variabilität enthalten darf.

Wäre ein Import zusätzliche Variabilitätspunkte möglich, müsste jedes mögliche Fragment mit einer separaten Variabilitätsbeschreibung exportiert werden und der Variabilitätsbeschreibung des Referenzprozesses zur Verfügung gestellt werden. Die Auswahl, welche

Fragmente exportiert werden müssen, könnte rekursiv bei der Ableitung der Fragmente getroffen werden.

Im Folgenden wird behandelt, wie sich die einzelnen Teile in einer Variabilitätsbeschreibung umsetzen lassen.

6.5.1. Variable Attribute

Variable Attribute werden als einzelne Variabilitätspunkte umgesetzt. Der Variabilitätspunkt eines variablen Attributs enthält einen XPATH-Ausdruck, der auf die Position des Attributs verweist. Da die Attribute im serialisierten XML sich nicht gesammelt an einer Stelle des Elements befinden, sind hier zahlreiche Sonderfälle nötig.

Die Alternativen der Attribute werden in Alternativen des Variabilitätspunkts umgesetzt. Freie Alternativen können dabei unverändert übernommen werden. Nur die Constraints müssen teilweise vorausgewertet werden, da die verwendeten Funktionen in der Variabilitätsbeschreibung nicht unterstützt werden (siehe auch Abschnitt 6.5.4). Explizite Alternativen müssen je nach Attribut in ein Element gehüllt werden, damit sie bei der Ableitung korrekt eingesetzt werden können. Der Grund dafür ist, dass manche Attribute im fertigen XML-Dokument als Element gespeichert werden, wie beispielsweise das Attribut *documentation*.

6.5.2. Variable Regionen

Eine variable Region wird bei der Ableitung wie oben beschrieben durch eines von mehreren Fragmenten ersetzt. Der Nutzer muss hierbei lediglich das Fragment wählen, weswegen die variable Region konzeptuell nur einen einzigen Variabilitätspunkt darstellt.

Für die Umsetzung der durch diesen Variabilitätspunkt beschriebenen Wahl sind allerdings mehrere Aktionen notwendig. In der Variabilitätsbeschreibung muss der konzeptuelle Variabilitätspunkt also in mehrere technische Variabilitätspunkte zerlegt werden, die die eigentliche Ableitung beschreiben.

Dies führt zu zwei Problemstellungen. Zum einen darf diese Zerlegung für den Nutzer der Variabilitätsbeschreibung nicht sichtbar sein. Zum anderen beziehen sich die Abhängigkeiten anderer Variabilitätspunkte auf diesen konzeptuellen Variabilitätspunkt.

Um diese beiden Probleme zu lösen, wird eine variable Region wie in Abbildung 6.11 in einzelne technische Variabilitätspunkte zerlegt. Die Darstellung orientiert sich dabei an der des Dependency-Graph. Einzelne Knoten sind technische Variabilitätspunkte, Kanten stellen Abhängigkeiten dar, durch die die Ableitungsreihenfolge festgelegt wird.

Der Grundgedanke dieser Zerlegung ist es, den Algorithmus in Listing 6.1 mit den Mitteln einer Variabilitätsbeschreibung auszudrücken.

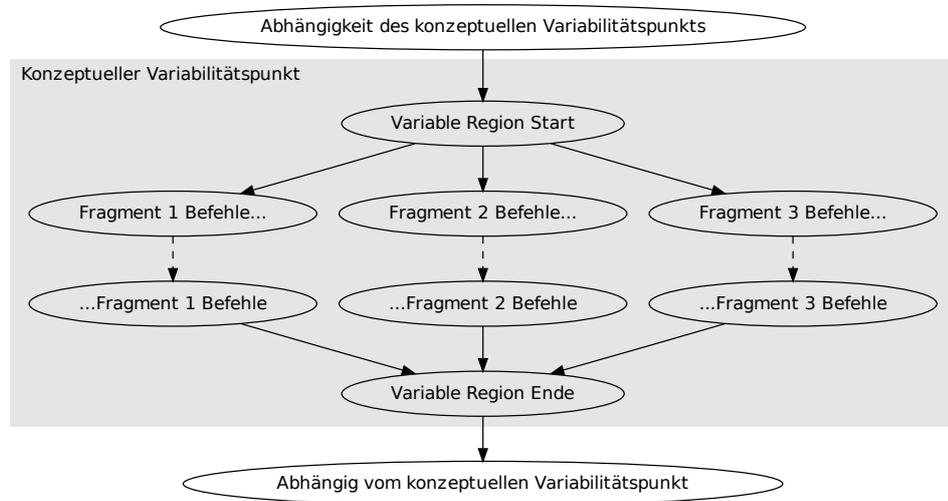


Abbildung 6.11: Export einer variablen Region

```

fragment = Fragmentauswahl();
switch (fragment) {
case Fragment1: setzeEin(Fragment1); break;
case Fragment2: setzeEin(Fragment2); break;
...
case FragmentX: setzeEin(FragmentX); break;
}
  
```

Listing 6.1: Algorithmus Fragmentwahl in der exportierten Variabilitätsbeschreibung

Die Variabilitätspunkte in Abbildung 6.11 entsprechen den einzelnen Befehlen dieses Algorithmus und werden im Folgenden näher beschrieben:

Variable Region Start Dieser Variabilitätspunkt enthält eine Alternative für jedes zur Wahl stehende Fragment mit der entsprechenden Enabling Condition. Er dient dazu, dem Nutzer die Wahl zu ermöglichen und hat dieselben Abhängigkeiten wie der konzeptuelle Variabilitätspunkt. Die getroffene Wahl hat keine unmittelbaren Auswirkungen auf das Dokument, wird aber später in Enabling Conditions verwendet.

Fragment X Befehle Für jedes einzusetzende Fragment wird eine Reihe von Variabilitätspunkten angelegt, mit denen die Einsetzung des jeweiligen Fragments vorgenommen wird, falls es in *Variable Region Start* gewählt wurde. Diese Variabilitätspunkte werden automatisch abgeleitet. Der Aufbau dieser Variabilitätspunkte wird im folgenden Abschnitt näher beschrieben.

Variable Region End Dieser Variabilitätspunkt steht stellvertretend dafür, dass die variable Region abgeleitet wurde. Er hat keine Änderungen des Dokuments zufolge und wird automatisch abgeleitet. Er wird als Dependency verwendet, falls andere Variabilitätspunkte von der variablen Region abhängig sind. Dafür ist er abhängig von den jeweils letzten Befehlen der einzelnen Fragmente. Durch die Transitivität wird so sichergestellt, dass Variabilitätspunkte, die von dem konzeptuellen Variabilitätspunkt abhängig sind, erst abgeleitet werden, wenn alle technischen Variabilitätspunkte abgeleitet wurden.

6.5.3. Fragmente

Im Rahmen des Exports einer variablen Region muss für die Alternativen jeweils ein Fragment behandelt werden. Ziel dieser Behandlung ist eine Reihe von technischen Variabilitätspunkten zur Fragmenteinsetzung. Abbildung 6.11 zeigt, wie diese Reihen in den konzeptuellen Variabilitätspunkt einzuordnen sind.

Diese Behandlung verläuft in drei Schritten: Vorbereitung, Umwandlung in XML und Fragmenteinsetzung.

Vorbereitung

Um diese Variabilitätspunkte zu erstellen, ist es notwendig, dass das Fragment als XML-Datei vorliegt. Wie bei der Ableitung im Client kann ein Fragment im JSON-Format angefordert werden. Allerdings ist ein Fragment noch kein gültiges BPMN 2.0, da es zusätzliche Elemente (Fragment-Start, -Link und -Ende) enthält.

Um dieses Problem zu beheben, wird im Vorbereitungsschritt das Fragment auf die Umwandlung vorbereitet. Dazu werden folgende Veränderungen vorgenommen:

Validierung des Fragments Das Fragment wird auf Gültigkeit (exakt ein Fragment-Start und -Ende, etc.) untersucht. Ist es nicht gültig, wird der variablen Region keine Alternative für dieses Fragment hinzugefügt.

Erweiterung der IDs Um sicherzustellen, dass die IDs der Elemente im Fragment innerhalb des Referenzprozesses einzigartig sind, werden sie mit einem Präfix erweitert, der für das Fragment und die variable Region einzigartig ist.

Entfernung von Fragment-Start und -Ende Da es sich bei Fragment-Start und -Ende nicht um gültige XML-Elemente handelt und sie zudem nicht in den Referenzprozess eingesetzt werden, können sie entfernt werden. Um später die ein- und ausgehenden Sequenzflüsse des Fragments setzen zu können, werden die entsprechenden Aufsetzpunkte zwischengespeichert.

Umwandlung der Fragment-Links Um den Export zu ermöglichen, werden Fragment-Links in gewöhnliche End-Events umgewandelt. Um sie später noch identifizieren zu können, wird eine Liste angelegt.

Umwandlung in XML

Mit Hilfe Oryx-eigener Funktionen wird das vorbereitete Dokument in BPMN 2.0 DI-XML umgewandelt. Mittels dieses umgewandelten Dokuments und der zwischengespeicherten Eigenschaften wird nun die Fragmenteinsetzung in der Variabilitätsbeschreibung umgesetzt.

Fragmenteinsetzung

Das Einsetzen eines Fragments wird in der Variabilitätsbeschreibung durch eine Reihe von Variabilitätspunkten umgesetzt.

Dazu enthalten diese Punkte jeweils zwei Alternativen. Eine enthält einen Teilbefehl (z.B. das Einsetzen eines Elements), die andere eine Aktion, die das Dokument nicht verändert. Mittels Enabling Conditions wird sichergestellt, dass die Alternative mit dem Teilbefehl genau dann ausgewählt wird, wenn der Nutzer das entsprechende Fragment gewählt hat. Hat er ein anderes Fragment gewählt, wird hingegen die Alternative ausgewählt, die das Dokument nicht verändert. So wird sichergestellt, dass nur das gewählte Fragment eingesetzt wird, ohne, dass der Nutzer eine weitere Entscheidung treffen muss. Um eine Ausführungsreihenfolge der Teilbefehle zu erzwingen, werden Dependencies verwendet.

Das Einsetzen eines Fragments setzt sich aus folgenden Variabilitätspunkten bzw. Schritten zusammen:

- 1. Elemente einsetzen** Diese Variabilitätspunkte dienen dazu, die BPMN-Elemente des Fragments in den Referenzprozess einzufügen.
- 2. Sequenzflüsse verbinden** Diese Variabilitätspunkte ändern Quelle bzw. Ziel des aus- bzw. eingehenden Sequenzflusses. Die Aufsetzpunkte im Fragment sind aus dem Vorbearbeitungsschritt bekannt.
- 3. Variable Region entfernen** Mit diesen Variabilitätspunkten wird die variable Region aus dem Referenzprozess entfernt.
- 4. Fragment-Links auflösen** Zur Auflösung der Fragment-Links wird zuerst das Fragment-Link-Element gelöscht und dann das Ziel aller eingehenden Flüsse (Sequenz- und Nachrichtenfluss) auf das Ziel des Links gesetzt.

6.5.4. Enabling Conditions

Da die Definition der Enabling Conditions auf Variabilitätsbeschreibungen basiert, ist nur eine Anpassung für den Export notwendig. Wie beim PreParser (siehe 6.4.3) müssen die definierten Funktionen ausgewertet werden, da sie zum Teil in der Variabilitätsbeschreibung nicht unterstützt werden.

Diese Auswertung wandelt die Funktionen in statische XPATH-Ausdrücke um, die dieselbe Funktion erfüllen wie ein Funktionsaufruf mit den entsprechenden Parametern.

6.5.5. Abhängigkeiten

Die Möglichkeiten zur Abhängigkeitsdefinition in der Oryx-Erweiterung decken sich mit denen der Variabilitätsbeschreibung. Die Abhängigkeiten können also unverändert übernommen werden.

Da eine variable Region allerdings wie beschrieben in mehrere Variabilitätspunkte exportiert wird, müssen die Abhängigkeiten jeweils zum richtigen dieser Variabilitätspunkte führen. Dies wird bewerkstelligt, indem ein Mapping mitgeführt wird, das den Namen einer variablen Region auf den entsprechenden Variabilitätspunkt abbildet, der als Abhängigkeit verwendet werden soll.

6.6. Deployment

Um einen späteren Einbau des Deployments zu vereinfachen, wurden zwei Plugins als Platzhalter implementiert, die den grundlegenden Deployment-Ablauf umsetzen. Geht man davon aus, dass eine fertig abgeleitete Variante dem Server übergeben wird, so ist es denkbar, von dort aus auch die weitere Verarbeitung durchzuführen. Möglich wäre beispielsweise, die Variante automatisch in einer Process Engine bereitzustellen und sie nach dem Deployment automatisch auszuführen.

Das clientseitige Deployment-Plugin überprüft zunächst, ob der Prozess noch Variabilität enthält. Ist dies der Fall, wird das Deployment dem Nutzer mit einer Fehlermeldung verweigert. So wird sichergestellt, dass nur vollständig abgeleitete Referenzprozesse für das Deployment an den Server gesendet werden.

Das serverseitige Deployment-Plugin erhält vom Client das Diagramm im JSON-Format. Da es sich nur um einen Platzhalter handelt, wird dieses nicht weiter bearbeitet. Eine Implementierung des Deployments kann hier anschließen.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Das Ziel dieser Arbeit war die Erstellung und Umsetzung eines Konzepts für Varianten in unternehmensübergreifenden Servicenetzwerken. Dieses wurde basierend auf den Praxisanforderungen und bisherigen Ansätzen konzipiert und implementiert. Für den Export der variantenreichen Prozesse wurde der Export in Variabilitätsbeschreibungen ermöglicht.

In Kapitel 2 wurden als Grundlage des Entwurfs Anforderungen aus dem Praxisprojekt *openXchange* abgeleitet.

Grundlagen zu den verwendeten Technologien wurden in Kapitel 3 behandelt. Unter anderem wurden die beim Export verwendeten Variabilitätsbeschreibungen (3.2) und der webbasierte Editor Oryx (3.3) vorgestellt, für den die Lösung implementiert wurde.

In Kapitel 4 wurden drei Ansätze zur Variabilität, PESOA, Prozesskonfigurator und Provop vorgestellt und miteinander verglichen.

Auf Basis dieser Vorarbeiten wurde in Kapitel 5 ein Variabilitätskonzept ermittelt, das Variabilitätsmodellierung (5.1), Abhängigkeitsmodellierung (5.2) und Variantenableitung (5.3) umfasst.

Die Variabilitätsmodellierung beinhaltet, welche Konstrukte für Variabilität zur Verfügung stehen und wie diese modelliert werden. Als grundlegende Struktur von Variabilität wurde zwischen Referenzprozessen und Prozessfragmenten differenziert. Referenzprozesse dienen als Grundgerüst für eine Prozessvariante. Prozessfragmente stellen einzelne Bauteile da, die in den Kontrollfluss des Referenzprozesses eingesetzt werden. Um diese beiden Diagrammtypen zu modellieren, wurden zusätzliche Modellierungselemente definiert. Die variable Region ist ein Platzhalter für ein einzusetzendes Fragment. Mit Fragment-Start und -Ende werden die Punkte markiert, an denen ein Fragment mit dem Sequenzfluss des Referenzprozesses verbunden wird. Fragment-Links dienen dazu, Prozess und Datenfluss auch über Fragmentgrenzen fortzusetzen. Attribute von BPMN-Elementen können ebenfalls variabel sein. Variable Attribute werden mittels einer eigenen Beschreibungssprache definiert.

Das Konzept zur Abhängigkeitsmodellierung definiert, welche Arten von Abhängigkeiten es zwischen einzelnen Variabilitätspunkten gibt. Die Dependencies eines Variabilitätspunkts

legen fest, welche Variabilitätspunkte abgeleitet werden müssen, bevor dieser Variabilitätspunkt abgeleitet werden kann. Sie erzwingen somit eine Ableitungsreihenfolge. Enabling Conditions dienen hingegen dazu, die Auswahl an Alternativen eines Variabilitätspunkts zu beschränken. Eine Alternative kann nur ausgewählt werden, wenn die Enabling Condition erfüllt ist.

Der Algorithmus für die Variantenableitung beschreibt, wie unter Beachtung der Abhängigkeiten eine Variante aus Referenzprozess und Fragmenten erstellt werden kann.

Für die Umsetzung dieses Variabilitätskonzept wurde der web-basierte Editor Oryx verwendet. Sie wird in Kapitel 6 beschrieben. Modellierungsmöglichkeiten für Referenzprozesse und Prozessfragmente wurden mittels Stencil Set Extensions hinzugefügt. Mittels Plugins können Abhängigkeiten und variable Attribute komfortabel erstellt und geändert werden. Ein weiteres Plugin ermöglicht die grafische Ableitung einer Variante. Der Export in eine Variabilitätsbeschreibung wurde als Server-Plugin umgesetzt. Rekursive Ableitung wurde im Export allerdings nicht berücksichtigt, da sie sich nicht mit den Mitteln der Variabilitätsbeschreibung ausdrücken lässt.

Abschließend soll in diesem Kapitel ein Ausblick über weitere Teilaspekte von Varianten in unternehmensübergreifenden Servicenetzwerken gegeben werden, die in dieser Arbeit nicht behandelt wurden. Für jedes dieser Gebiete werden mögliche Wege zur Umsetzung skizziert. Die in dieser Arbeit erstellten Konzepte und Software dienen dabei als Grundlage.

7.2. Verteilte Modellierung

Während Oryx als Plattform die gemeinsame Bearbeitung von Referenzprozessen und Fragmenten ermöglicht, ist es wünschenswert, die Modellierung und Ableitung auch simultan durchführen zu können.

ProcessWave.org[WWWh] ist ein Projekt des Hasso-Plattner-Instituts, in dem ein kollaboratives Modellierungswerkzeug für Google Wave entwickelt wird. Google Wave[WWWj] ist eine Plattform für Echtzeitkollaboration über das Internet.

Dieser Processwave-Editor basiert auf Oryx, bietet aber weniger Features. Mit ihm können Diagramme in Echtzeit von mehreren Personen modelliert werden.

Eine Untersuchung des Quellcodes[WWWi] ergibt, dass sowohl Plugins als auch Stencilsets wie bei Oryx verwendet werden. Unklar ist, inwieweit existierende Plugins angepasst werden müssen und ob alle Features von Oryx unterstützt werden. Es sollte aber möglich sein, die im Rahmen dieser Arbeit entwickelten Plugins zu portieren.

Im Gegensatz zu Oryx enthält der ProcessWave-Editor allerdings kein Backend. Die Speicherung wird über Google Wave erledigt und ermöglicht einen Export zu einer existierenden

Oryx-Instanz. Für das Fragment-Repository und den Export müssten also andere Lösungen gefunden werden.

Leider werden weder Google Wave noch der ProcessWave-Editor momentan weiterentwickelt. Wegen mangelnden Erfolges wird die Google Wave-Plattform zum Ende des Jahres 2010 abgeschaltet, wird aber eventuell als Open Source-Software veröffentlicht[Hö].

7.3. Variantenausführung

Geht man davon aus, dass eine fertig abgeleitete Variante per Server-Plugin gespeichert wird, so ist es denkbar, von dort aus auch die weitere Verarbeitung durchzuführen. Möglich wäre beispielsweise, die Variante in einer Process Engine bereitzustellen und sie nach dem Deployment automatisch auszuführen. Da sich eine Variante allerdings als gültiges BPMN 2.0 XML-Diagramm exportieren lässt, ist eine solche Integration zwar wünschenswert, aber nicht zwingend notwendig. Es kann also eine beliebige BPMN 2.0-fähige Process Engine verwendet werden.

Process Engines für BPMN 2.0 stehen zum Zeitpunkt dieser Arbeit noch am Anfang. Ein vielversprechender Ansatz ist Activiti[WWWo], das neben einem Oryx-basierten Editor auch eine Process Engine enthält. Editor und Engine sind über den sogenannten Activiti Cycle miteinander verbunden, der eine Integration zwischen Editor und Process Engine ermöglicht.[WWWd]

Activiti unterstützt allerdings noch nicht den vollen Umfang der BPMN 2.0.[WWWp] Beispielsweise werden Pools und Lanes noch nicht unterstützt.

Die Process Engine jBPM[WWWe] bietet ebenfalls Ansätze zur BPMN 2.0-Ausführung, die in einem Proof Of Concept vorgestellt wurden[Bar]. Die Unterstützung ist allerdings begrenzt und umfasst lediglich die Elemente Start- und End-Event sowie Task und Human-Task. Durch den Weggang der führenden Entwickler zu Activiti ist außerdem fraglich, wie die Weiterentwicklung verläuft.[Gui]

Vergleicht man die beiden Engines, bietet Activiti zum Zeitpunkt dieser Arbeit sowohl die umfangreichere BPMN 2.0-Unterstützung als auch die besseren Entwicklungsaussichten.

7.4. Optimierung

Die vorliegende Arbeit ermöglicht ein Mass Customizing von Geschäftsprozessen. In Verbindung mit Monitoring- und Qualitätskonzepten können diese Varianten in Hinsicht auf Performance, Kundenzufriedenheit und ähnliche Kriterien verglichen werden, gegebenenfalls auch unternehmensübergreifend.

Dies ermöglicht es, eine Optimierung der Varianten vorzunehmen. Wie bereits beschrieben kann eine Variante mittels einer Variabilitätsbeschreibung auch automatisiert abgeleitet werden. Denkbar ist ein auf evolutionären Algorithmen basierender Ansatz zur Prozessoptimierung.

Evolutionäre Algorithmen dienen der Optimierung und basieren auf Erkenntnissen der Biologie. Wie in der biologischen Evolution wird hier auf ein Überleben des Stärksten gezielt. Schlechte Lösungen „sterben“, gute „überleben“.

Ein evolutionärer Algorithmus beginnt mit einer Anfangsmenge von Lösungen, der sogenannten Initialpopulation, und führt auf dieser Basis eine Evolution durch. Dazu verwendet ein evolutionärer Algorithmus folgenden Kreislauf [Wil]:

- 1. Mutation** Vorhandene Lösungen werden mit einer kleinen Chance in Details verändert. Bei einer Prozessvariante kann dies z.B. ein Grenzwert sein. Diese veränderten Lösungen werden der Population hinzugefügt.
- 2. Kreuzung** Zwei Lösungen werden miteinander gekreuzt, d.h. ihre Eigenschaften werden nach dem Zufallsprinzip miteinander rekombiniert. Bei Zahlenwerten ist auch ein Durchschnitt oder Ähnliches denkbar. Die aus der Kreuzung resultierenden Lösungen werden ebenfalls der Population hinzugefügt.
- 3. Selektion** Jede Lösung in der Population wird hinsichtlich ihrer Qualität bewertet. Die besten Lösungen werden für den nächsten Durchlauf ausgewählt. Der Rest wird verworfen, sodass die Anzahl der Lösungen wieder der zu Beginn des Durchlaufs entspricht.

Dieser Zyklus wird so lange wiederholt, bis eine akzeptable Lösung gefunden wurde.

Wendet man dieses Konzept auf die Variantenbildung an, so ist es denkbar, dass man manuell erstellte Varianten als Anfangspopulation wählt.

In der Mutationsphase kann man basierend auf den vorhandenen Varianten neue erstellen, wobei darauf zu achten ist, dass die Constraints und Abhängigkeitsbeziehungen erfüllt sind, also nur gültige Mutationen entstehen. So ist beispielsweise denkbar, dass ein Schwellenwert für eine Einholung von mehreren Angeboten variiert wird, so lange er sich innerhalb der Constraints bewegt.

Bei der Rekombination müssen ebenfalls illegale Varianten verhindert werden. Dies lässt sich beispielsweise durch Backtracking bei der zufälligen Entscheidung, von welcher Elternvariante eine Entscheidung gewählt wird, erreichen.

Für die Selektionsphase gibt es zwei Möglichkeiten, zum einen Simulation, zum anderen ein Feldversuch. Ersteres hat den Vorteil, dass eine rekombinierte Variante nicht in den

Produktivbetrieb entlassen wird. Dem steht aber der Nachteil gegenüber, dass Simulationsergebnisse nur bedingt repräsentativ sind und sich für Kriterien wie beispielsweise Kundenzufriedenheit nur schwerlich erstellen lassen.

Der Feldversuch ist zeitaufwändiger, da er über eine angemessene Anzahl von Prozessdurchläufen durchgeführt werden muss. Außerdem birgt er das Risiko, einem Kunden einen suboptimalen Prozess zu liefern. Hier ist besonders wichtig, dass jede gültige Variante auch eine Variante ist, deren Ausführung prinzipiell akzeptabel ist.

Die im Versuchszeitraum anfallenden Prozessdurchführungen werden gleichmäßig über die Varianten der Population verteilt und die erhobenen Qualitätsdaten separat gespeichert. Nachdem eine gewisse Anzahl von Durchgängen abgeschlossen wurde, kann die Selektion vorgenommen werden. Die Prozesse mit der schlechtesten Qualität werden verworfen, der Rest der Population beginnt den nächsten Zyklus.

Ist die Bereitschaft, mutierte und gekreuzte Prozesse im Produktivbetrieb zu verwenden, nicht vorhanden, lassen sich für die verschiedenen Phasen auch Alternativen denken. Für die Mutations- und Selektionsphase kann anstatt einer automatischen Erstellung neuer Prozesse auch eine manuelle vorgenommen werden. Denkbar wäre hier, eine Möglichkeit zum paarweisen Vergleich von Prozessen bei gleichzeitiger Anzeige der Qualitätsunterschiede zu ermöglichen. So könnte ein Anwender beide Prozesse miteinander kombinieren und so ein neues Individuum erstellen.

Soll hingegen der Selektionsschritt nicht automatisiert stattfinden, ist auch hier eine Ersetzung durch eine manuelle Auswahl denkbar. So kann beispielsweise auch der Algorithmus abgebrochen werden, wenn man einen Prozess gefunden hat, mit dem man zufrieden ist. Diese manuellen Elemente dürften die Laufzeit der evolutionären Optimierung nicht stark verändern, da der Feldtest der Varianten der limitierende Zeitfaktor ist.

Möchte ein Teilnehmer einen weniger unberechenbaren Prozess, so ist es auch möglich, die evolutionäre Optimierung nur auf einem kleinen Teil der Prozessdurchläufe rechnen zu lassen. Der Rest könnte dann mit einer manuell gewählten Variante durchgeführt werden.

7.5. Simulation

In Verbindung mit dem Variantenkonzept bietet es sich an, Simulationen mit fertigen Varianten durchzuführen, um zu überprüfen, ob die Ableitung korrekt verlaufen ist und um verschiedene Varianten miteinander zu vergleichen.

Im einfachsten Fall bedeutet solch eine Simulation einen einfachen Prozessdurchlauf noch in der Modellierungsumgebung, mit dem ein grober Funktionstest durchgeführt wird. Oryx bietet bereits Möglichkeiten für diese Art von Simulation. Ein sogenannter *Stepthrough* ist

unter Anderem für BPMN 1.2 implementiert[NS]. Für diesen Stepthrough wird der BPMN-Prozess intern in ein Petri-Netz umgewandelt. Eine Umsetzung für BPMN 2.0 ist zwar geplant, aber leider noch nicht vorhanden[WK].

Im Bereich der variantenreichen Prozesse gibt es aber auch andere Einsatzgebiete für Simulation. Im vorhergehenden Abschnitt wurde ein Konzept zur Prozessoptimierung beschrieben, das ebenfalls eine Simulationskomponente verwenden könnte, um einzelne Varianten miteinander zu vergleichen. Vorteile gegenüber einem Feldtest ist zum einen wie oben beschrieben, dass die Variante nicht im Produktivbetrieb getestet werden muss, und zum anderen, dass eine Simulation schneller Ergebnisse liefert als ein Feldversuch.

Einige Hersteller bieten *Business Process Simulation*-Werkzeuge für verschiedene Prozessbeschreibungssprachen an[JvNo6]. Da aber bei älteren BPMN-Versionen kein einheitliches Dateiformat existiert, können diese oft nur Prozesse simulieren, die mit dem Werkzeug selbst modelliert wurden.

OXProS[GBDo6] ist ein Projekt zur Entwicklung einer Process Simulation Engine für BPMN. Allerdings kann mit OXProS momentan erst ein Teil der BPMN simuliert werden. Ebenso wie die vorliegende Arbeit ist OXProS als Erweiterung in Oryx integriert.

Im Oryx-Client können BPMN-Prozesse um Parameter für die Simulation erweitert werden, beispielsweise zur Verzögerung bei der Ausführung von Tasks. Der Editor sendet das erweiterte Diagramm im XPDL-Format an den Server, der dann die Simulation vornimmt. Auch bei OXProS werden BPMN-Prozesse für die Simulation zu Petri-Netzen umgewandelt[GBDo6]. Ergebnis der Simulation ist ein Log im MXML-Format. Dieses Log muss vom Nutzer ausgewertet werden, da OXProS keine eigenen Auswertungsfunktionen mitbringt.

Dank der Einbindung in Oryx ist es denkbar, das Simulations- und Variantenkonzept miteinander zu verbinden. Referenzprozess und Fragmente müssten dann vom Nutzer um das Simulationsverhalten erweitert werden. Jede entstehende Variante hätte somit alle notwendigen Daten, um simuliert zu werden. Dies könnte beispielsweise im Rahmen der im vorherigen Abschnitt skizzierten Optimierung Verwendung finden. Anstatt die einzelnen zu vergleichenden Varianten im Selektionsschritt in das Produktivsystem zu verlassen, könnten sie im Simulator geprüft werden. Die einzelnen Vergleichsdaten müssten dann aus den Logdateien der Durchläufe gewonnen werden.

Gibt es Kriterien, die sich per Simulation nicht ermitteln lassen, ist es auch denkbar, ein hybrides Verfahren zu verwenden. Entweder werden Simulation und Feldtest parallel verwendet, um verschiedene Messwerte zu gewinnen oder man schaltet beides hintereinander. Im ersten Schritten werden die Varianten mit dem Simulator überprüft. Danach kann entweder eine Positiv-Auswahl von Varianten getroffen werden oder einzelne Varianten können durch bestimmte KO-Kriterien ausgeschlossen werden. Die übrigen Varianten können dann im Feldtest genauer untersucht werden.

7.6. Validierung und Compliance

Durch das im Rahmen dieser Arbeit entwickelte Abhängigkeitskonzept ist bereits eine grundlegende Validierung der gebildeten Variante während der Ableitung möglich. Mittels Enabling Conditions lassen sich so beispielsweise Nachrichtenflüsse mit nichtvorhandenem Ziel und Ähnliches vermeiden.

Da die Modellierung von Referenzprozess und Fragmenten durch verschiedene Personen durchgeführt werden kann, ist ein hoher Abstimmungsaufwand vonnöten, diese verschiedenen Teile miteinander konsistent zu halten. So kann eine Umbenennung von Elementen im Referenzprozess zur Folge haben, dass Constraints und Enabling Conditions in den Fragmenten nicht mehr ordnungsgemäß funktionieren. Es wäre wünschenswert, solch eine enge Kopplung über Diagrammgrenzen hinweg zu vermeiden.

Um eine allgemeine Validierung der abgeleiteten Variante zu ermöglichen, ist es denkbar, dem Referenzprozess Validierungsregeln hinzuzufügen. Diese könnten am Ende des Ableitungsvorgangs ausgewertet werden, um zu überprüfen, ob die Ableitung korrekt verlaufen ist. Es bietet sich an, dafür ähnlich wie bei den Enabling Conditions Xpath mit den zusätzlichen Funktionen zu verwenden. Allerdings müssten diese erweitert werden, um notwendige Validierungsregeln auszudrücken.

Beispiele für Anfragen, die umsetzbar sein müssen sind, ob Flüsse korrekt sind (innerhalb desselben Pools, gültiger Start bzw. Ende, etc.) oder wie Elemente im Fluss zueinander stehen (vorher, nachher, nebenläufig, etc.).

BPMN-Q

BPMN-Q[WWW1] ist eine grafische Anfragesprache für BPMN-Diagramme, mit der die oben genannten Anfragen umgesetzt werden könnten[Awa07].

Anfragen in BPMN-Q werden grafisch als Diagramme modelliert. Sie können entweder auf einen einzelnen BPMN-Prozess angewendet werden, um einen Wahrheitswert zu erhalten, oder auf eine Menge von Modellen, um diejenigen zu erhalten, auf die die Anfrage zutrifft. Außerdem ist es möglich, Anfragen an einen gerade laufenden oder bereits ausgeführten Prozess zu stellen, beispielsweise, um Deadlocks zu erkennen[Awa07].

Abbildung 7.1 zeigt ein einfaches Beispiel für solche Anfrage. Sie ist dann erfüllt, wenn sich zwischen *Alternative A* und *Alternative B* kein unmittelbarer Sequenzfluss befindet. Analog ließe sich auch mit einer anderen Beziehung abfragen, ob es einen indirekten Sequenzfluss zwischen den beiden Aktivitäten gibt. Dies ist z.B. dazu gut, um zu prüfen, ob sich zwei Aktivitäten gegenseitig ausschließen.

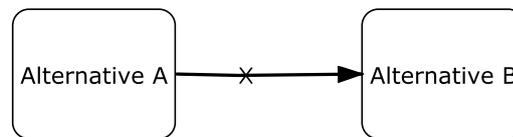


Abbildung 7.1.: Beispiel für eine BPMN-Q-Anfrage. (Eigene Darstellung)

Der Einsatz von BPMN-Q würde sich für das vorliegende Projekt aus zweierlei Gründen anbieten. Zum einen ist die grafische Modellierung darauf ausgelegt, sowohl für Fach- als auch Geschäftsanwender verständlich zu sein[Awa07]. Zum anderen existiert bereits eine Implementierung von BPMN-Q für Oryx[ADWo8].

Compliance-Deskriptoren

Eng verbunden mit dem Problem der Validierung ist das der Compliance. Aufgrund gesetzlicher und anderer Anforderungen müssen Prozesse heutzutage eine Vielzahl von Normen und Richtlinien einhalten[SALMo9].

Es gibt mehrere Ansätze, diese Einhaltung zu erzwingen. So können Compliance-Regeln beispielsweise während der Prozessausführung erzwungen oder jeder einzelne Prozess nach der Modellierung manuell (z.B. mittels Review) auf Compliance überprüft werden. Letztere Option sorgt bei Prozessvarianten allerdings für einen unzumutbaren Aufwand, da jede einzelne entstandene Variante überprüft werden müsste.

Wünschenswert wäre daher ein Ansatz, der es erlaubt, die Einhaltung von Compliance schon während der Modellierung bzw. während der Ableitung zu erzwingen. In [SALMo9] werden sogenannte *Compliance-Deskriptoren* vorgestellt, die genau dies ermöglichen. Analog zu Variabilitätsbeschreibungen speichern sie separat vom eigentlichen Prozess Compliance-Regeln. Der Prozess ist in diesem Fall abstrakt, d.h. er muss ähnlich zu dem in dieser Arbeit vorgestellten Referenzprozess erst erweitert werden, bevor eine konkrete Variante entsteht.

Der Ansatz in [SALMo9] ist sprachunabhängig und lässt sich somit auch auf BPMN anwenden. Exemplarisch wird dort BPEL verwendet. Im Folgenden soll eine Integration von Compliance-Deskriptoren und dem vorliegenden Variabilitätskonzept skizziert werden.

Ein abstrakter Geschäftsprozess in BPEL enthält zwei Arten von Aktivitäten:

Opaque Activities Sogenannte *Opaque Activities* sind ein Teil von BPEL[WWWm] und dienen dazu, einen Prozess später mit zusätzlichen Elementen zu erweitern.

Compliance Activities Sogenannte *Compliance Activities* dienen dazu, bestimmte Aktivitäten vor Benutzeränderungen zu schützen. Sie dürfen bei der Modellierung der Variante weder entfernt noch geändert werden. Ein Beispiel für solche Aktivitäten sind vom Gesetzgeber vorgeschriebene Prüfungen.

Überträgt man diese Konzepte auf den vorgestellten Ansatz, so lassen sie sich wie folgt abbilden: *Opaque Activities* entsprechen den variablen Regionen, denn beide dienen dazu, zusätzliche Elemente in den abstrakten Prozess bzw. Referenzprozess einzusetzen. *Compliance Activities* sind gewissermaßen schreibgeschützt. Im vorliegenden Ansatz darf der Referenzprozess ebenfalls bei der Ableitung nicht verändert werden. Allerdings kann die Variante von Hand geändert werden, wenn auch der Referenzprozess nur von seinem Besitzer überschrieben wird. Um dies zu verhindern, wäre es nötig, entweder wie in [SALM09] vorgeschlagen einen Schreibschutz bei der Modellierung umzusetzen oder aber beim Öffnen des Referenzprozesses augenblicklich die Ableitung zu beginnen, in der das Diagramm außer durch Wahl einer Alternative nicht geändert werden kann.

Im Compliance-Deskriptor werden sogenannte *Compliance Points* definiert, die sich aus zwei Teilen zusammensetzen:

Compliance Links Sogenannte *Compliance Links* definieren, welche Aktivitäten im Prozess *Compliance Activities* sind.

Compliance Assurance Rules *Compliance Assurance Rules* sind die eigentlichen Compliance-Regeln. Sie sind mit einer oder mehreren *Opaque Activities* verbunden, für die sie gelten. Um sicherzustellen, dass sie erfüllt sind, werden sie auf die einzusetzenden Alternativen angewandt. Nur Alternativen, die alle mit einer *Opaque Activity* verbundenen Regeln erfüllen, dürfen eingesetzt werden.

Da das vorgestellte Variantenkonzept keine freie Modellierung für die Variantenerstellung benötigt, müssen *Compliance Activities* nicht gesondert ausgewiesen werden. Wie oben beschrieben kann stattdessen angenommen werden, dass alle Elemente, die keine Variabilität enthalten, *Compliance Activities* sind. Allerdings ist diese Kategorisierung eventuell nicht fein genug. So dürfen z.B. bei Elementen des Referenzprozesses, die variable Attribute enthalten, nur diese Attribute verändert werden, der Rest der Aktivität bzw. ihr Platz im Prozess allerdings nicht.

Compliance Assurance Rules ähneln konzeptionell *Enabling Condition*, insofern, dass sie die Auswahl an Alternativen beschränken. Denkbar ist es, die *Compliance Assurance Rules* wie zusätzliche *Enabling Conditions* zu behandeln und an derselben Stelle zu überprüfen. Damit eine Alternative gewählt werden kann, müsste also sowohl ihre eventuell vorhandene *Enabling Condition* erfüllt sein als auch alle ihrer variablen Region zugehörigen *Compliance Assurance Rules*.

Mit den sprachlichen Mitteln zur Beschreibung von Enabling Conditions lassen sich diese Regeln allerdings nicht umsetzen. Enabling Conditions betrachten immer den Referenzprozess, während Compliance Assurance Rules sich auf das einzusetzende Fragment beziehen. Beispiele für Compliance Assurance Rules sind das Verbot, im eingesetzten Fragment den Prozess zu beenden, bestimmte Compliance Activities mit Links zu umgehen oder bestimmte Daten des Referenzprozesses zu manipulieren.

Für die Umsetzung der Compliance Assurance Rules gibt es noch keine konkrete Modellierungssprache. In [SALMo9] werden Regeln lediglich mittels Formeln und Pseudocode dargestellt. Es bietet sich an, die oben vorgestellte BPMN-Q zu verwenden, da sie die Untersuchung von Fragmenten auf die oben vorgestellten Kriterien hin ermöglicht. Zur Überprüfung der Regel wird diese dann als BPMN-Q-Anfrage auf das infragekommende Fragment angewendet.

Bei der rekursiven Einsetzung von Fragmenten (z.B. wenn ein Fragment seinerseits eine variable Region enthält) ist es fraglich, welche Regeln anwendbar sein sollen. Zum einen ist es denkbar, dass das Fragment seinerseits Compliance-Anforderungen an seine variablen Regionen stellt, zum anderen muss aber auch die Compliance aus Sicht des Referenzprozesses gewahrt bleiben, da das rekursiv eingesetzte Fragment letzten Endes in eine variable Region des Referenzprozesses eingesetzt wird.

Die Regeln lassen sich aber unter Umständen nicht einfach übertragen. Nimmt man beispielsweise eine Kardinalitätsregel, die besagt, dass ein bestimmtes Element nur exakt einmal im Fragment enthalten sein darf, das die variable Region ersetzt, so sorgt die rekursive Anwendung der Regel für Fehler. Denn das solcherart beschränkte Element kann entweder im Fragment oder in einem der einzusetzenden Unterfragmente enthalten sein, damit die Regel erfüllt ist. In diesem Fall kann also gesehen auf die einzelne Einsetzung gar nicht entschieden werden, ob die Regel erfüllt ist oder nicht, denn das fehlende Element könnte noch immer durch eine rekursive Einsetzung hinzukommen.

Eine Möglichkeit, diese Probleme zu umgehen, wäre es, das einzusetzende Fragment vor der Einsetzung so weit abzuleiten, dass es keine Variabilität mehr enthält. Dann könnte das abgeleitete Fragment ein einziges Mal auf Compliance untersucht werden. Dies hätte allerdings für den Nutzer den Nachteil, dass ihm ein Fehler erst mitgeteilt wird, wenn er das Fragment vollständig abgeleitet hat.

Ebenso wie die Variabilitätsbeschreibung wird der Compliance-Deskriptor als separates Dokument gespeichert. Dies müsste im vorliegenden Werkzeug beim Export berücksichtigt werden.

A. Anhang

A.1. Inhalt und Aufbau des beigelegten Datenträgers

Der beigelegte Datenträger ist wie folgt aufgebaut:

Ausarbeitung Der Ordner `ausarbeitung` enthält dieses Dokument.

Projektverzeichnis Das Eclipse-Projekt, das die vorgenommenen Erweiterungen an Oryx enthält findet sich im Ordner `projekt`.

Prototyp Der fertig kompilierte Prototyp findet sich im Verzeichnis `distribution`.

Tests Die Testberichte, Testbeschreibungen und Testdaten befinden sich im Ordner `test`. außerdem ist das verwendete Testwerkzeug *TOM* enthalten.

SVN-Patch Der Ordner `patch` enthält einen Patch, mit dem sich die Erweiterungen in einen aktuellen SVN-Checkout von Oryx einpflegen lassen.

Installationsdateien Im Ordner `install` finden sich Installationsdateien für alle Software, die für den Betrieb der Lösung notwendig ist. Es sind nur Installationsdateien für Windows enthalten.

Videos Die im Client zur Verfügung gestellten Videoanleitungen sind im Ordner `videos` enthalten.

A.2. Installation

Im Folgenden wird die Installation einer Entwicklungsumgebung beschrieben, in der die vorliegende Lösung weiterverwendet werden kann. Dabei wird davon ausgegangen, dass ein Windows-System eingesetzt wird. Unter Linux muss prinzipiell dieselbe Software installiert werden, was sich je nach Distribution / Paketverwaltung anders erledigen lässt. Das vorliegende Setup wurde ebenfalls unter Ubuntu 10.04 getestet.

Folgende Software muss installiert werden:

- Aktuelles Java JDK

- Firefox mit Firebug
- Python
- PostgreSQL
- Apache Tomcat
- Eclipse

Sie können entweder die Installationsdateien auf dem beiliegenden Datenträger verwenden oder aktuelle Versionen von den Seiten der Hersteller herunterladen. Beachten Sie allerdings, dass es insbesondere zwischen Python und PostgreSQL zu Inkompatibilitäten kommen kann.

Wollen Sie keine Entwicklungsumgebung aufsetzen, sondern nur die vorliegende Lösung nutzen, können Sie sich die Installation von Eclipse und Firebug sparen. Kopieren Sie nach der Installation einfach die mitgelieferten Dateien `oryx.war` und `backend.war` in den `/webapps`-Ordner Ihrer Tomcat-Installation.

Installation des Java JDK

Das aktuelle Java JDK findet sich unter <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Starten Sie den Installer und folgen Sie den Anweisungen auf dem Bildschirm. Sie müssen an den Voreinstellungen nichts ändern.

Ist das JDK installiert, muss es noch in die Umgebungsvariablen eingetragen werden. Unter Windows XP finden Sie die Umgebungsvariablen beispielsweise mittels Rechtsklick auf den Arbeitsplatz → Eigenschaften → Erweitert → Umgebungsvariablen.

Fügen Sie hier eine neue Systemvariable namens `JAVA_HOME` hinzu, die den Pfad zum JDK enthält (z.B. `C:\Programme\Java\jdk1.6.0_22`). Außerdem muss die `PATH`-Variable um den `/bin`-Ordner des JDK erweitert werden (z.B. `C:\Programme\Java\jdk1.6\bin`).

Installation von Firefox mit Firebug

Firefox können Sie unter <http://www.mozilla-europe.org/de/firefox/> herunterladen.

Folgen Sie den Anweisungen des Installers. Es sind keine besonderen Einstellungen notwendig.

Starten Sie Firefox und besuchen Sie die Seite <http://getfirebug.com/>, um Firebug zu installieren. Klicken Sie dort die Schaltfläche „Install Firebug for Firefox“. Alternativ können Sie die vorhandene Installationsdatei verwenden.



Abbildung A.1.: Installation von Firebug

Es sollte sich ein Dialog wie in Abbildung A.1 öffnen. Folgen Sie den Anweisungen auf dem Bildschirm. Nach dem Neustart von Firefox sollte Firebug zur Verfügung stehen. Sie erkennen dies an einem Käfer-Icon in der unteren rechten Ecke des Fensters.

Installation von Python

Die aktuelle Version von Python können Sie unter <http://www.python.org/download/> herunterladen. Beachten Sie allerdings, dass es zwischen Python und PostgreSQL zu Kompatibilitätsproblemen kommen kann.

Installieren Sie Python mittels des Installers. Es sind keine besonderen Einstellungen vonnöten.

Da PostgreSQL in der vorliegenden Version nur mit Python 2.5 zusammenarbeitet, muss ihm diese Version vorgetauscht werden. Gehen Sie dazu in den /System32-Ordner Ihrer Windows Installation (für gewöhnlich C:\Windows\System32). Kopieren Sie dort die Datei `python26.dll` und benennen Sie sie in `python25.dll` um.

Installation von PostgreSQL

Sie können PostgreSQL unter <http://www.postgresql.org/download/> herunterladen. Bitte beachten Sie die Kompatibilität mit der Version von Oryx, die Sie verwenden wollen.

Installieren Sie PostgreSQL mit Hilfe des Installers. Merken Sie sich die Passwörter für den angelegten Nutzer `postgres` und die Datenbank. Sie werden sie später brauchen. Bei dem Auswahl-dialog zu den Sprachen müssen Sie einen Haken bei Python setzen. Ist dies nicht möglich, so bedeutet dies, dass Python nicht ordnungsgemäß installiert wurde. Sie können die Installation in diesem Fall nicht erfolgreich abschließen, bevor sie nicht Python korrekt installieren.

Das `/bin`-Verzeichnis der Postgres-Installation (z.B. `C:\Programme\PostgreSQL\8.3\bin`) muss in die Umgebungsvariable `Path` eingetragen werden. Unter Windows XP finden Sie die Umgebungsvariablen beispielsweise mittels Rechtsklick auf den Arbeitsplatz → Eigenschaften → Erweitert → Umgebungsvariablen.

Nachdem die Datenbank installiert ist, muss sie noch für die Verwendung von Oryx aufgesetzt werden. Öffnen Sie dazu die Windows-Command-Line (Start → Ausführen → `cmd`) und geben Sie folgenden Befehl ein:

```
runas /user:postgres cmd
```

In der sich nun öffnenden Command-Line sind Sie der Nutzer `postgres`. Falls Sie um ein Passwort gebeten werden, verwenden Sie das, das Sie bei der Installation von PostgreSQL angegeben haben.

Geben Sie nun folgenden Befehl ein, um den Nutzer `poem` anzulegen. Wählen Sie als Passwort ebenfalls `poem`. Bei den Fragen, die Ihnen gestellt werden, gilt folgendes: Der Nutzer ist kein Superuser, darf Datenbanken erstellen, aber keine neuen Rollen. Abbildung A.2 zeigt, welche Werte Sie in dem Dialog eingeben müssen.

```
createuser -U postgres --echo --pwprompt --encrypted poem
```

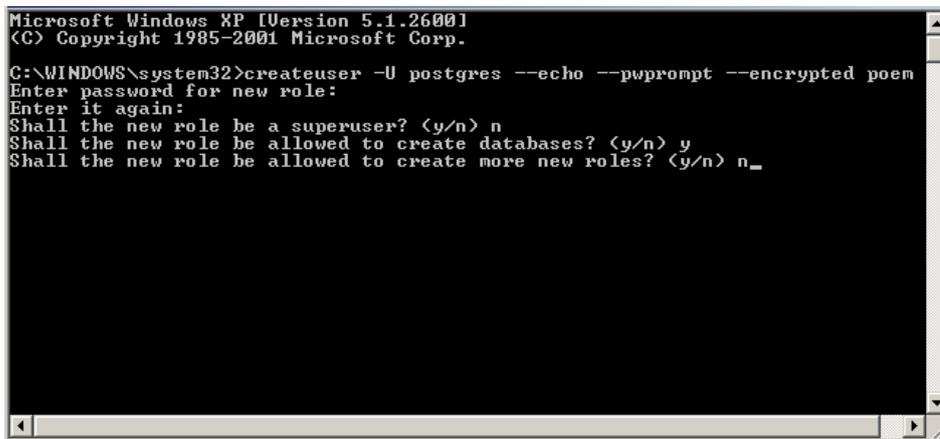
Legen Sie nun die Datenbank für Oryx an:

```
createdb -U postgres --echo --encoding utf8 --owner poem poem
```

Nun muss noch die Schema-Datei geladen werden, damit die Datenbank alle nötigen Tabellen enthält. Setzen Sie im unteren Befehl den Pfad zu der mitgelieferten `db_schema.sql` ein. Wollen Sie die Tests durchführen, können Sie stattdessen die Datei `test.sql` aus dem Test-Ordner verwenden.

```
psql -U postgres --dbname poem --file <Pfad und Name der Schema-Datei>
```

Falls kein Fehler auftritt, ist damit das Aufsetzen der Datenbank abgeschlossen.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>createuser -U postgres --echo --pwprompt --encrypted poem
Enter password for new role:
Enter it again:
Shall the new role be a superuser? <y/n> n
Shall the new role be allowed to create databases? <y/n> y
Shall the new role be allowed to create more new roles? <y/n> n_
```

Abbildung A.2.: Anlegen des Nutzers Poem

Installation von Apache Tomcat

Die aktuellste Version von Apache Tomcat finden Sie unter <http://tomcat.apache.org/>.

Installieren Sie Tomcat mittels des Installers. Beachten Sie, dass der Pfad des JDK angegeben werden muss, obwohl nach einem JRE gefragt wird. Ein JRE reicht nicht aus. Merken Sie sich das Passwort.

Tomcat kann mittels der Datei `tomcat6w.exe` im `/bin`-Verzeichnis der Installation gestartet werden.

Installation von Eclipse

Entpacken Sie Eclipse in einen Ordner Ihrer Wahl. Öffnen Sie die Datei `eclipse.ini` und ändern Sie den Eintrag `-Xmx`, sodass er `-Xmx1024m` lautet. Damit erhöhen Sie den verfügbaren Arbeitsspeicher für Eclipse. Bei kleineren Werten könnte es später zu Abstürzen kommen.

Starten Sie nun Eclipse. Für die Verwendung von SVN müssen nun die nötigen Plugins installiert werden. Wählen Sie dafür im Menü Help den Eintrag „Install new Software“.

Wählen Sie nun die Hauptupdateseite als Quelle (z.B. Helios für Eclipse Helios). Wählen Sie in der nun erscheinenden List unter Collaboration die Einträge „Subversive SVN Integration“ und „Subversive Team Provider“. Abbildung A.3 zeigt, welche Einträge Sie auswählen müssen. Klicken Sie nun auf next und folgen Sie den Anweisungen des Installers.

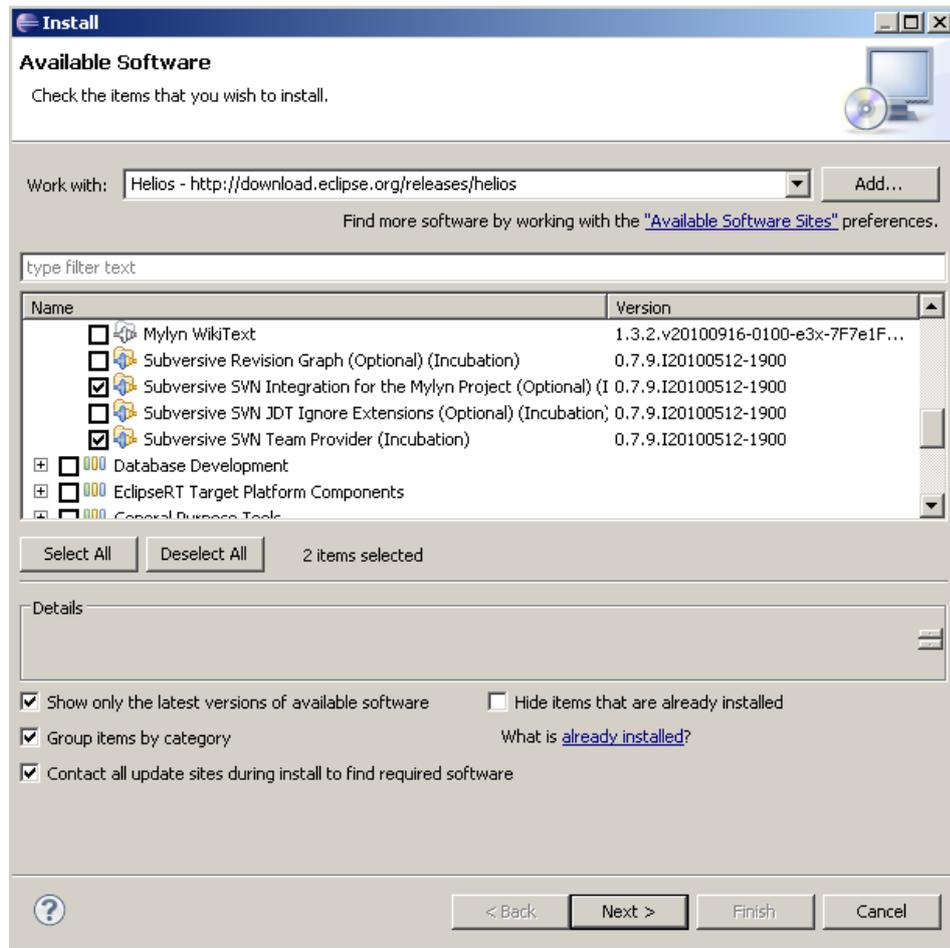


Abbildung A.3.: Installation des SVN-Plugins für Eclipse

Nach dem Neustart von Eclipse wählen Sie die Option „SVNKit 1.3.2“. Es folgt ein weiterer Installationsdialog. Danach sind alle notwendigen Plugins installiert.

Einrichten des Projekts

Grundsätzlich haben Sie zwei Möglichkeiten, die vorliegende Lösung in Oryx zu verwenden. Zum einen können Sie das gesamte Eclipse-Projekt in ihren Workspace-Ordner kopieren und öffnen.

Zum anderen können Sie eine aktuelle Version von Oryx aus dem Repository laden und die Änderungen mit einem Patch einpflegen. Bei dieser Vorgehensweise sind allerdings eventuell weitere Anpassungen nötig, falls Versionskonflikte auftreten.

Um Oryx auszuchecken gehen Sie wie folgt vor: Wählen Sie File→New→Other→SVN→Project from SVN aus.

Geben Sie als Adresse <http://oryx-editor.googlecode.com/svn/trunk> ein. Unter dem Reiter Advanced müssen Sie „Structure recognition“ ausschalten. Nun können Sie den Checkout vornehmen. Falls es mit der Head-Revision nicht funktioniert, wählen Sie von Hand die aktuellste Revision. Geben Sie als Projektname `oryx` ein und folgen Sie den Anweisungen. Das Herunterladen des Repository kann selbst mit einer schnellen Internet-Verbindung eine Weile dauern.

Haben Sie das Projekt heruntergeladen, können Sie den Patch mittels Rechtsklick auf den Ordner `oryx`→Team→Apply Patch anwenden. Wählen Sie die Patch-Datei aus und bestätigen Sie mit next. Im darauffolgenden Dialog werden Probleme angezeigt, die eventuell beim Anwenden des Patches auftreten. Sie müssen manuell behoben werden.

Da es sich um Binärdateien handelt, stellt der Patch keinerlei Icons wieder her. Dies betrifft folgende Ordner, in die Sie die Icons manuell einfügen müssen:

/oryx/editor/data/stencilsets/extensions/bpmn2.0fragment Sie finden die Icons im Unterordner `fragment` des Patches.

/oryx/editor/data/stencilsets/extensions/bpmn2.0variable Sie finden die Icons im Unterordner `variable` des Patches.

/oryx/editor/client/images Sie finden die Icons im Unterordner `images` des Patches.

Build einrichten

Um Oryx aus Eclipse heraus kompilieren und aufsetzen zu können müssen Sie zuerst das `/webapps`-Verzeichnis Ihrer Tomcat-Installation unter `deploymentdir` in die `build.properties`-Datei im Hauptverzeichnis des Projekts eintragen. Beachten Sie dabei, dass Sie die Backslashes escapen müssen (aus jedem `\` wird ein `\\`).

Nun können Sie eine neue External Tools-Configuration anlegen. Sie finde diese unter dem Punkt „External Tools“ in der Werkzeugleiste. Wählen Sie für die neue Konfiguration einen neuen Ant Build aus. Als Buildfile wählen Sie die Datei `build.xml` aus (z.B. `${workspace_loc:/oryx/build.xml}`). Als Basisverzeichnis wählen Sie das Hauptverzeichnis des Projekts (z.B. `${workspace_loc:/oryx}`). Im Reiter Targets wählen Sie `build-all` und `deploy-all`, um das Projekt zu kompilieren und Tomcat zur Verfügung zu stellen. Abbildung A.4 zeigt die korrekte Konfiguration.

Mittels Run können Sie den Build-Prozess starten. Falls Sie alles richtig gemacht haben, sollten zwei Dateien `oryx.war` und `backend.war` erstellt und in den `/webapps`-Ordner Ihres Tomcat kopiert werden. Tomcat installiert die neuen Archive dann automatisch und stellt sie unter `http://localhost:8080/backend/poem/repository` zur Verfügung.

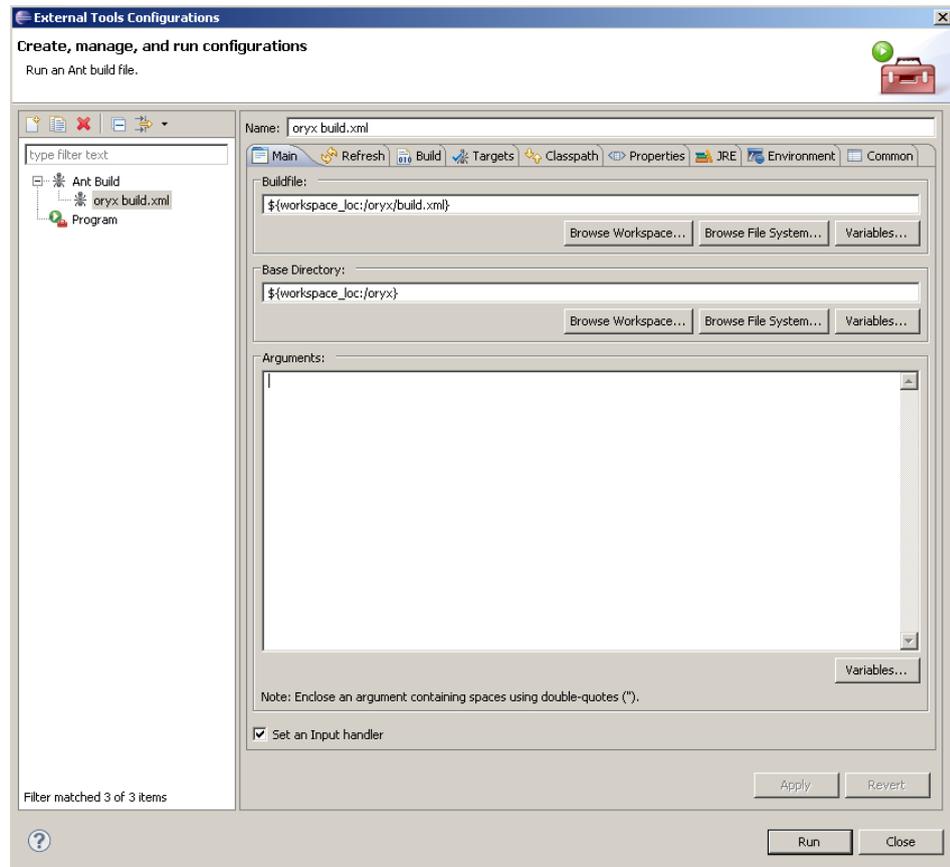


Abbildung A.4.: Konfiguration des Build-Prozesses in Eclipse

Falls dies nicht geschieht, können Sie versuchen, Tomcat neu zu starten, um das Deployment manuell anzustoßen.

Falls das Kompilieren einen Fehler hervorruft, können Sie versuchen, in der Datei `build.xml` bei `build-all` den Eintrag `generate-build-dependency-graph` zu löschen. Er hat bei einer Testinstallation zu Problemen geführt und ist für das Kompilieren nicht notwendig.

A.3. Anleitung

Im Folgenden wird eine kurze Anleitung gegeben, wie die Erweiterung verwendet werden kann. Es wird davon ausgegangen, dass der Anwender grundlegende Oryx-Kenntnisse hat. Die Begriffe und Screenshots basieren auf englischen Spracheinstellungen. Für die

deutsche Spracheinstellung sind sie anders, entsprechen aber den in der vorliegenden Arbeit gebrauchten.

Unter `videos` auf dem beigelegten Datenträger finden Sie zusätzlich Video-Tutorials, die die einzelnen Aspekte der Erweiterung anschaulich erklären.

Übersicht



Abbildung A.5.: Variabilitätswerkzeuge

In der Oryx-Werkzeuggestreife finden Sie die einzelnen Werkzeuge der Erweiterung. In Abbildung A.5 sehen Sie von links nach rechts:

Ableitungsmodus ein/ausschalten Dient dazu, eine Variante abzuleiten.

Dependency-Graph anzeigen Zeigt den serialisierten Dependency-Graph an.

Variability Wizard Dient der Definition von Dependencies, variablen Attributen und wählbaren Fragmenten.

Ableitungsprotokoll Zeigt ein Protokoll aller vorgenommenen Ableitungsentscheidungen an.

Fragmentanzeige Hebt die eingesetzten Fragmente hervor.

Hilfe Abruf der Tutorial-Videos.

Variabilitätsbeschreibung anzeigen Exportiert den Referenzprozess in eine Variabilitätsbeschreibung und zeigt sie an.

Variabilitätsbeschreibung herunterladen Exportiert den Referenzprozess in eine Variabilitätsbeschreibung und bietet sie zum Herunterladen an.

Deployment Platzhalter für das Deployment. Überprüft, ob noch Variabilität im Diagramm vorhanden ist.

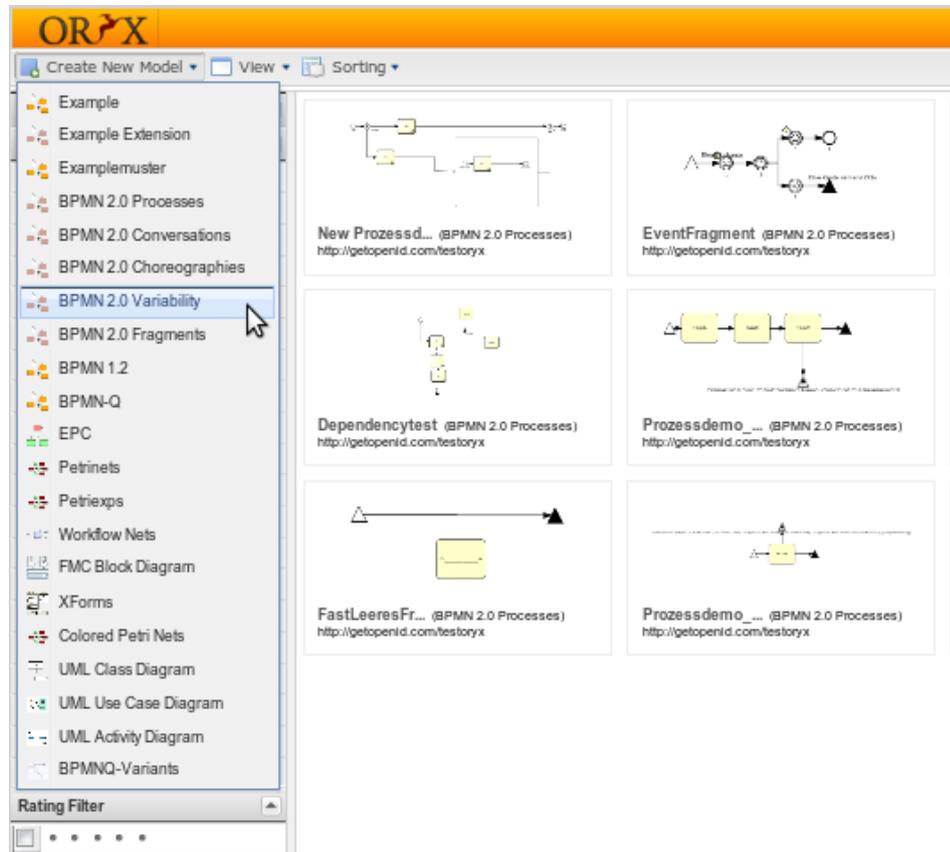


Abbildung A.6.: Erstellung eines neuen Diagramms

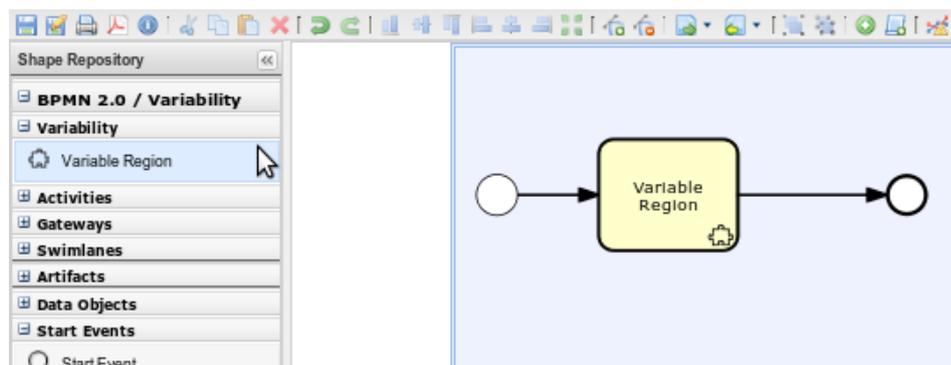


Abbildung A.7.: Variable Region im Shape Repository und im Diagramm

Referenzprozessmodellierung

Der Referenzprozess dient als Rahmen für die entstehende Variante. Er kann variable Regionen und Attribute enthalten. Um einen Referenzprozess zu modellieren, erstellen Sie ein neues Diagramm mit dem Stencil Set *BPMN 2.0 Variability* (siehe Abbildung A.6).

Sie können den Referenzprozess modellieren wie einen gewöhnlichen BPMN 2.0-Prozess. Zusätzlich dazu steht Ihnen die variable Region unter der Kategorie *Variability* zur Verfügung (siehe Abbildung A.7). Eine variable Region dient als Platzhalter für ein einzusetzendes Fragment und muss genau einen ein- und ausgehenden Prozessfluss haben.

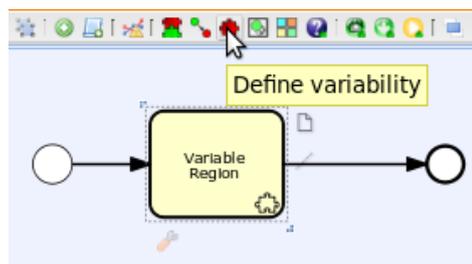


Abbildung A.8.: Aufruf des Variability Wizard

Um festzulegen, durch welche Fragmente die variable Region ersetzt werden darf, markieren Sie die Region und rufen Sie wie in Abbildung A.8 dargestellt den Variability Wizard auf.

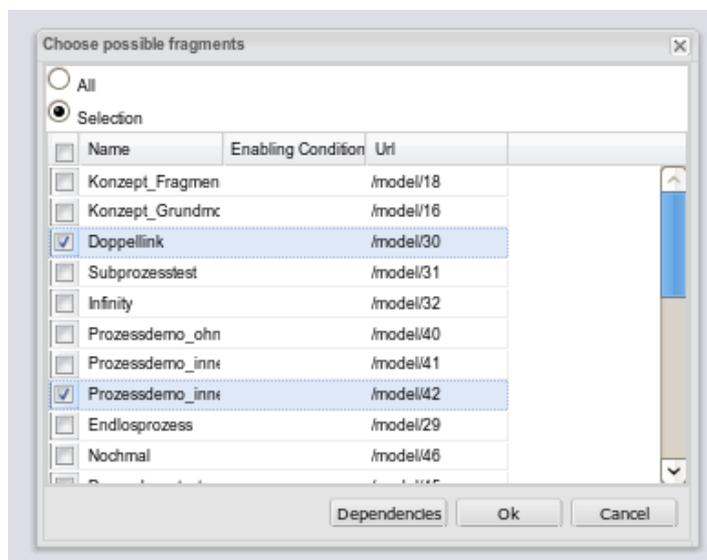


Abbildung A.9.: Auswahl der möglichen Fragmente

Im sich nun öffnenden Fenster (siehe Abbildung A.9) können Sie festlegen, welche Fragmente zur Wahl stehen dürfen. Treffen Sie keine konkrete Wahl, sind alle Fragmente erlaubt, die demjenigen zur Verfügung stehen, der die Variantenableitung vornimmt. Dies können auch Fragmente sein, die zur Erstellung des Referenzprozesses noch gar nicht existieren. So ist es auch möglich, Fragmente mehrerer Nutzer im selben Referenzprozess zu verwenden. Sie können im Oryx-Repository Fragmente entweder öffentlich oder für einzelne andere Nutzer sichtbar machen.

Bei der Angabe spezifischer Fragmente können Sie sog. *Enabling Conditions* definieren, die die Wählbarkeit des Fragments von einer Bedingung abhängig machen. Mehr dazu erfahren Sie weiter unten.

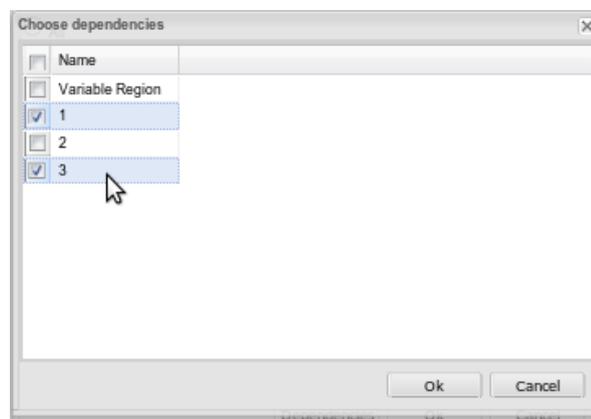


Abbildung A.10.: Definition von Dependencies

Möchten Sie Dependencies zu anderen Variabilitätspunkten festlegen, können Sie dies mittels der Schaltfläche *Dependencies* tun (siehe Abbildung A.10). Achten Sie darauf, keine zyklischen Dependencies festzulegen.

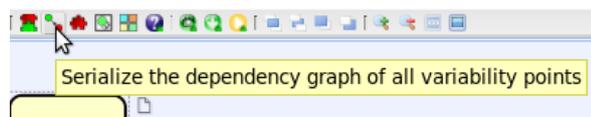


Abbildung A.11.: Anzeigen des Dependency-Graph

Um zu überprüfen, wie die Dependencies des Prozesses aussehen, können Sie sich den Dependency-Graph anzeigen lassen. Rufen Sie dazu wie in Abbildung A.11 gezeigt das entsprechende Plugin auf.

Es öffnet sich eine Serialisierung des Dependency-Graph, die Sie mittels des verlinkten Online-Renderers anzeigen lassen können (siehe Abbildung A.12). Ein Pfeil von A nach B

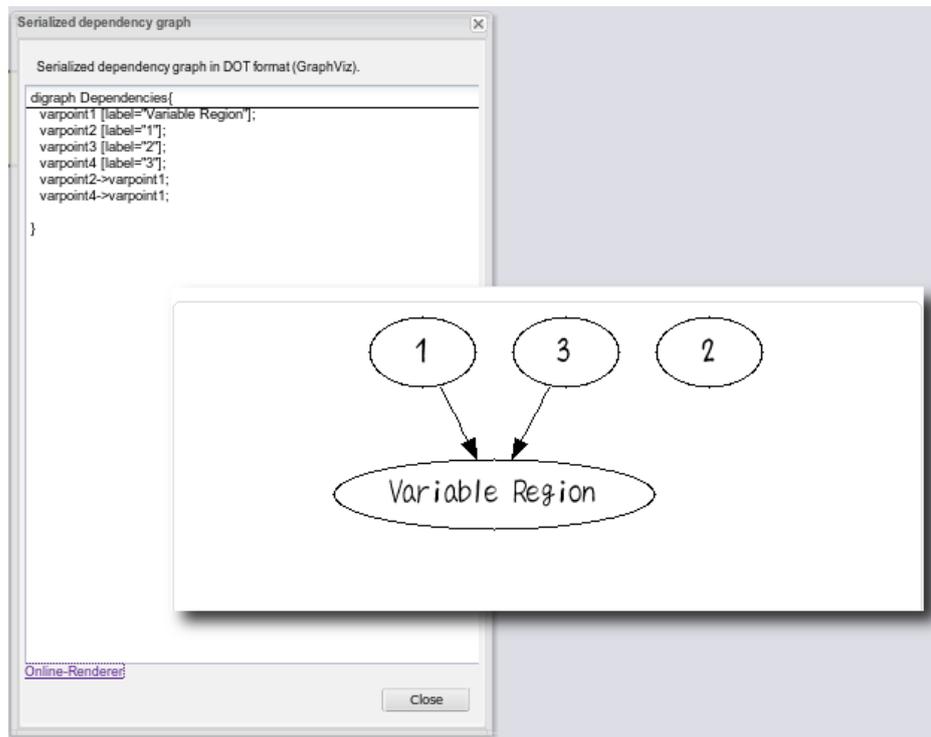


Abbildung A.12.: Serialisierter Dependency-Graph

bedeutet dabei, dass B von A abhängig ist. Damit ein Variabilitätspunkt abgeleitet werden kann, müssen vorher alle Punkte abgeleitet werden, von denen er abhängig ist. Variable Attribute sind im Graph nach Elementen gegliedert.

Variable Attribute

Um variable Attribute festzulegen, wählen Sie das Element, das das gewünschte Attribut enthält und öffnen Sie den Variability Wizard.

Es öffnet sich ein Fenster, in dem Sie das gewünschte variable Attribut auswählen können. Bei einigen Elementen wird allerdings nur eine Vorauswahl der gebräuchlichsten Attribute angezeigt. Um alle Elemente zu sehen, betätigen Sie den Button *Show all*.

Mit einem Doppelklick können Sie ein variables Attribut definieren oder modifizieren.

Der sich nun öffnende Dialog (siehe A.14) bietet Ihnen einen Überblick über das variable Attribut. Das es genau wie eine variable Region ein Variabilitätspunkt ist, können Sie mittels der Schaltfläche *Dependencies* Abhängigkeiten zu anderen Variabilitätspunkten festlegen.

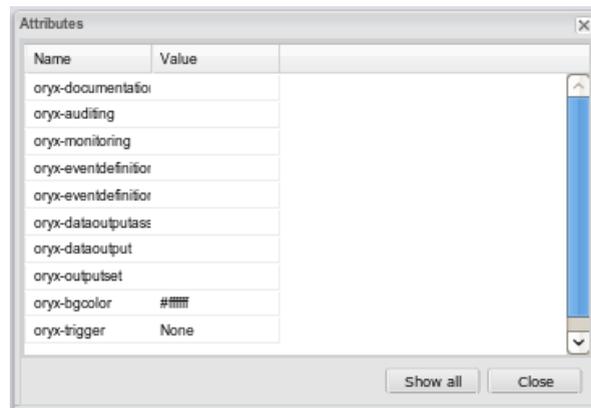


Abbildung A.13.: Auswahl variabler Attribute

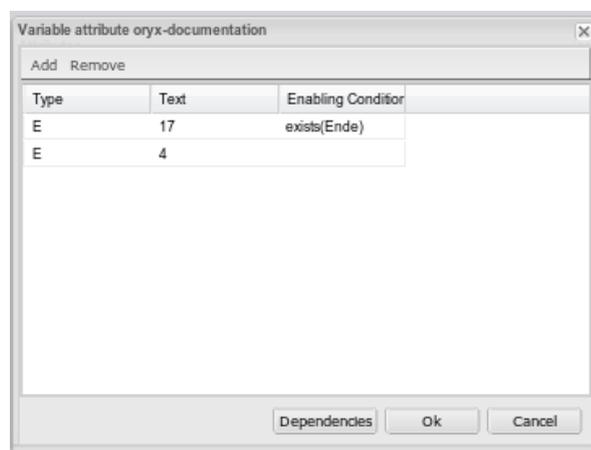


Abbildung A.14.: Editieren eines variablen Attributs

Mittels *Add* und *Remove* können Sie einzelne Alternativen hinzufügen und entfernen. Existierende Alternativen können Sie direkt in der Tabelle editieren.

Die Erweiterung unterscheidet zwischen zwei Arten von Alternativen:

Explizite Alternative Die explizite Alternative ermöglicht es, einen fest vorgegebenen Text zur Auswahl zu stellen.

Freie Alternative Die freie Alternative ermöglicht es, den Nutzer bei der Variantenableitung einen Text eingeben zu lassen. Dieser kann mittels eines Constraints auf Bedingungen überprüft werden.

Zusätzlich kann die Wahl jeder Alternative von einer sogenannten *Enabling Condition* abhängig gemacht werden. Die Alternative darf nur gewählt werden, wenn die Enabling Condition erfüllt ist.

Um Constraints und Enabling Conditions zu definieren, wird die *XML Path Language* (XPATH) verwendet¹. Zusätzlich zur Syntax von XPATH können folgende Funktionen verwendet werden:

\$value Ein Platzhalter für den vom Nutzer eingegebenen Wert. Kann nur in Constraints verwendet werden.

exists (ElementName) Mittels dieser Funktion kann überprüft werden, ob ein Element im Diagramm existiert. Als Parameter wird der Name des Elements ohne Anführungszeichen übergeben.

variabilityPoint (VariabilityPointName) Wählt einen Variabilitätspunkt anhand des Namens aus. Verwenden Sie diese Funktion, falls ein Variabilitätspunkt als Parameter einer Funktion verlangt wird.

selectedValue (VariabilityPoint) Diese Funktion gibt die gewählte Alternative eines Variabilitätspunkts als String zurück. Bei variablen Regionen ist dies die Url des Fragments, bei Attributen der Wert des Attributs. Wurde noch kein Wert gewählt, wird „undefined“ zurückgegeben.

elementAttribute (ElementName, AttributeName) Gibt den Wert eines Attributs als String zurück. Existiert das Element oder das Attribut nicht, wird „undefined“ zurückgegeben.

Im Folgenden werden ein paar Beispiele aufgelistet, um die Erstellung zu vereinfachen.

```
$value <= 100 and $value > 0
```

Dieser Constraint gibt an, dass ein Wert zwischen 1 und 100 eingegeben werden muss.

```
exists (Zahlungsabwicklung)
```

Diese Enabling Condition ist genau dann wahr, wenn im Diagramm ein Element namens „Zahlungsabwicklung“ existiert.

```
selectedValue (variabilityPoint (element#oryx-documentation)) =  
elementAttribute (element, oryx-documentation)
```

¹<http://www.w3.org/TR/xpath/>

Die linke Seite der Gleichung gibt den gewählten Wert eines Variabilitätspunkts an, der ein variables Attribut ist. Die rechte Seite der Gleichung gibt den momentanen Wert des variablen Attributs an. Diese Enabling Condition ist also immer wahr, wenn das variable Attribut bereits abgeleitet wurde.

Prozessfragmente

Prozessfragmente dienen dazu, in den Referenzprozess eingesetzt zu werden. Sie können Prozessfragmente modellieren, indem Sie den Diagrammtyp *BPMN 2.0 Fragments* wählen.

Ebenso wie Referenzprozesse können Prozessfragmente variable Attribute und Regionen enthalten. Bei der Fragmenteinsetzung wird diese Variabilität dann abgeleitet (mehr dazu unter A.3). Zusätzlich dazu stehen drei weitere Elemente zur Verfügung: *Fragment-Start*, *Fragment-Ende* und *Fragment-Link*.

Ein Prozessfragment hat genau einen Fragment-Start und ein Fragment-Ende. Diese beiden Elemente dienen als Verbindungsstellen zum Referenzprozess. Das Fragment wird so eingesetzt, dass der eingehende Fluss der zu ersetzenden variablen Region dort endet, wo der Fragment-Start sich befindet. Der ausgehende Fluss der Region beginnt dann dort, wo das Fragment-Ende sich befindet.

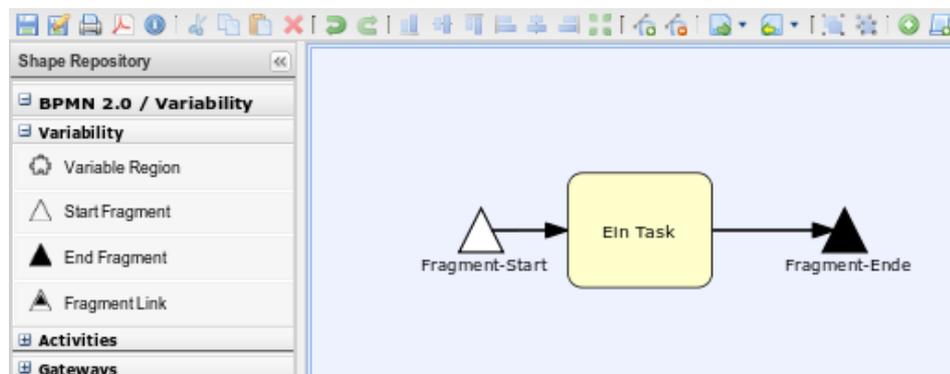


Abbildung A.15.: Ein einfaches Prozessfragment

Abbildung A.15 zeigt ein einfaches Prozessfragment. Setzt man es ein, wird die variable Region durch einen einfachen Task ersetzt.

Der Fragment-Link dient dazu, weitere Prozess- oder Nachrichtenflüsse aus dem Prozessfragment in den Referenzprozess zu ermöglichen. Dazu können Sie wie in Abbildung A.16 den entsprechenden Fluss mit dem Fragment-Link verbinden. Mittels des Attributs *target* können Sie den Namen des Elements angeben, mit dem der Fluss bei der Ableitung verbunden

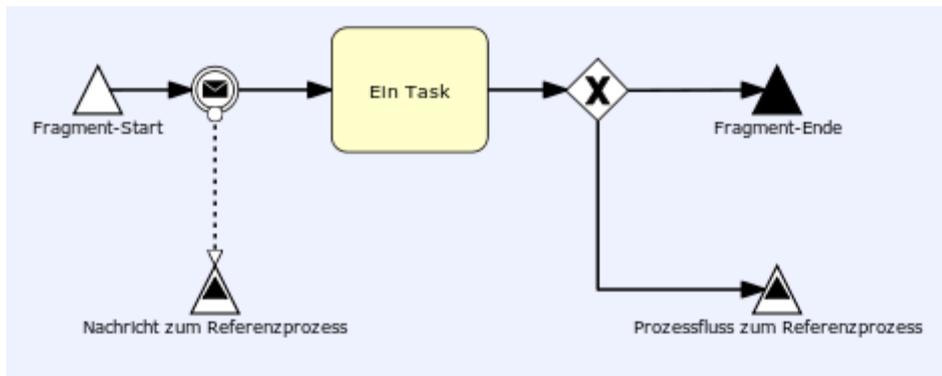


Abbildung A.16.: Fragment-Links für Prozess- und Nachrichtenfluss

werden soll. Selbstverständlich können Sie auch hier ein variables Attribut verwenden, um das Ziel des Links wählbar zu machen.

Falls Sie innerhalb des Fragments Constraints und Enabling Conditions verwenden, können Sie Bezug auf Elemente und Variabilitätspunkte des Referenzprozesses nehmen. Dabei müssen Sie allerdings darauf achten, dass es zu Fehlern kommen kann, wenn diese Elemente bei der Ableitung nicht gefunden werden können.

Variantenableitung

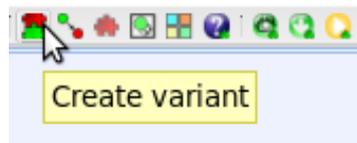


Abbildung A.17.: Starten der Ableitung

Haben Sie den Referenzprozess und alle notwendigen Fragmente modelliert, können Sie eine Variante ableiten. Versetzen Sie dazu das Diagramm in den Ableitungsmodus (siehe Abbildung A.17). Im Ableitungsmodus ist das Diagramm für Änderungen gesperrt und die variablen Teile werden eingefärbt.

Die Elemente, die bereits abgeleitet werden können, werden grün eingefärbt. Sind die *Dependencies* eines Variabilitätspunktes noch nicht erfüllt, wird er rot eingefärbt und kann noch nicht abgeleitet werden. Im Ableitungsmodus werden die Variabilitätspunkte nun unter Beachtung der definierten Dependencies und Enabling Conditions sukzessive abgeleitet, bis keine Variabilität im Diagramm mehr vorhanden ist. Während der Ableitung darf das Diagramm nicht gespeichert werden.

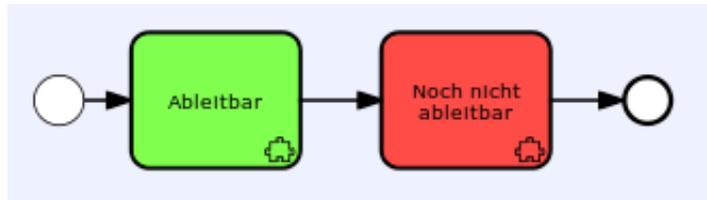


Abbildung A.18.: Markierte Elemente

Wählen Sie ein Element mit Variabilität aus, erscheint auf der rechten Seite eine Liste von Alternativen, die sich je nach Art der Variabilität unterscheidet. Im Folgenden werden die einzelnen Arten behandelt.

Variable Regionen

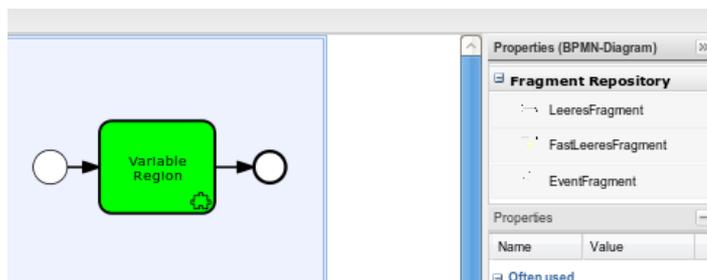


Abbildung A.19.: Ableitung einer variablen Region

Wählen Sie eine variable Region, erscheint wie in Abbildung A.19 eine Liste von Fragmenten. Wählen Sie eines davon aus und es wird eingesetzt. Enthält das Fragment seinerseits Variabilität, so muss diese vollständig abgeleitet werden, bevor die Ableitung des Referenzprozesses fortgesetzt werden kann. Das Plugin ändert in diesem Fall automatisch die Markierungen, sodass Sie stets wissen, wo Sie die Ableitung fortsetzen können.

Variable Attribute

Die Ableitung eines variablen Attributs verläuft ähnlich. Enthält ein Element mehrere variable Attribute, deren Dependencies erfüllt sind, so müssen Sie zuerst auswählen, welches der beiden Attribute Sie ableiten möchten.

Ebenso wie bei der Ableitung einer variablen Region erscheint an der rechten Seite eine Liste mit Alternativen (siehe A.20). Es werden selbstverständlich nur solche Alternativen angezeigt, deren Enabling Condition erfüllt ist.

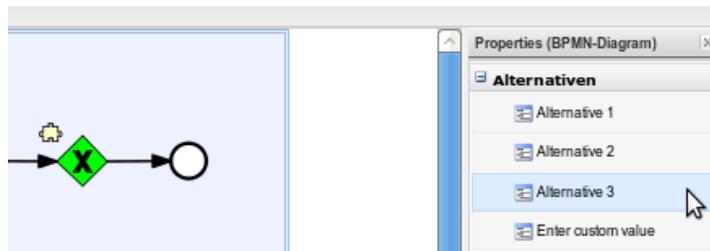


Abbildung A.20.: Ableitung eines variablen Attributs

Wählen Sie eine freie Alternative, können Sie einen eigenen Wert eingeben, der dann auf den Constraint der Alternative überprüft wird. Ist der Constraint nicht erfüllt, müssen Sie ihre Eingabe ändern.

Variable Fragment-Links

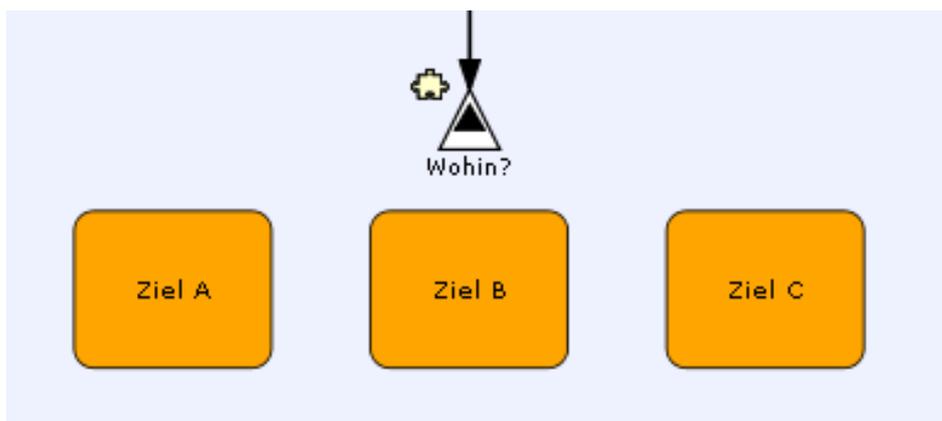


Abbildung A.21.: Zielwahl für einen Fragment-Link

Haben Sie einen Fragment-Link mit variablem Ziel definiert, können Sie dies bei der Ableitung direkt im Diagramm ableiten, anstatt einer Alternative aus der Liste zu wählen. Dabei werden alle möglichen Ziele wie in Abbildung A.21 markiert und Sie müssen nur noch das richtige wählen.

Automatische Ableitung

Gibt es z.B. durch Enabling Conditions nur eine einzige Möglichkeit, wie ein Variabilitätspunkt abgeleitet werden kann, so nimmt das Plugin diese Ableitung automatisch vor. Dies

kann auch gezielt genutzt werden, um eine einzelne Entscheidung (beispielsweise, ob ein Feature genutzt werden soll) an mehreren Stellen im Prozess umzusetzen.

Ableitungsprotokoll

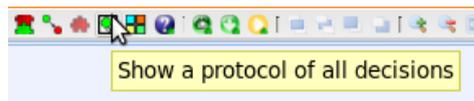


Abbildung A.22.: Aufruf des Ableitungsprotokolls

Sie können jederzeit ein Protokoll der getroffenen Entscheidungen einsehen. Rufen Sie es wie in Abbildung A.22 dargestellt auf.

Anzeige der eingesetzten Fragmente

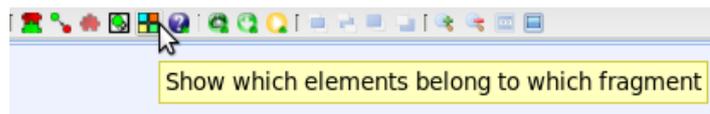


Abbildung A.23.: Anzeige der eingesetzten Fragmente einschalten

Haben Sie die Ableitung abgeschlossen, wird der Ableitungsmodus automatisch beendet. Möchten Sie nun sehen, welche Teile des Referenzprozesses aus welchen Fragmenten eingesetzt wurde, können Sie wie in Abbildung A.23 dargestellt einen Modus zur Fragmentanzeige aktivieren.

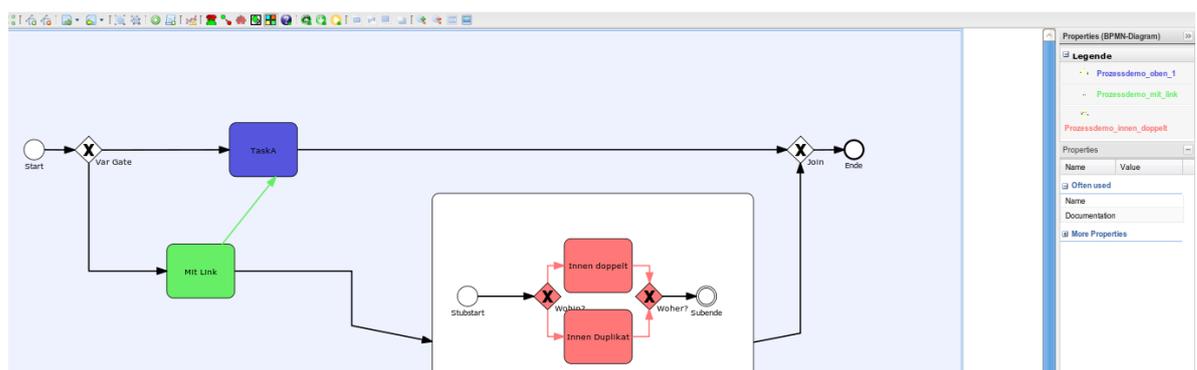


Abbildung A.24.: Eingefärbte Fragmente

In diesem Modus werden die eingesetzten Fragmente in verschiedenen Farben eingefärbt. In der rechten Seitenleiste finden Sie eine Legende, die den Farben die Fragmente zuordnet. Abbildung A.24 zeigt ein Beispiel für einen eingefärbten Referenzprozess.

Export

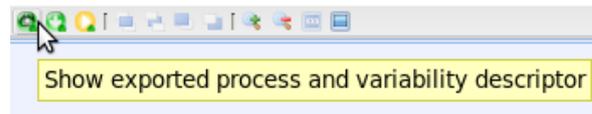


Abbildung A.25.: Referenzprozess exportieren und anzeigen

Sie können einen Referenzprozess auch als XML-Datei mit zugehöriger Variabilitätsbeschreibung exportieren. Beachten Sie dabei aber, dass die einsetzbaren Fragmente keine Variabilität enthalten dürfen. Beim Export haben Sie die Wahl, sich die exportierten Dateien anzusehen oder sie herunterzuladen, wie in Abbildung A.25 zu sehen. Rechts daneben befindet sich das Icon zum Download des exportierten Prozesses.

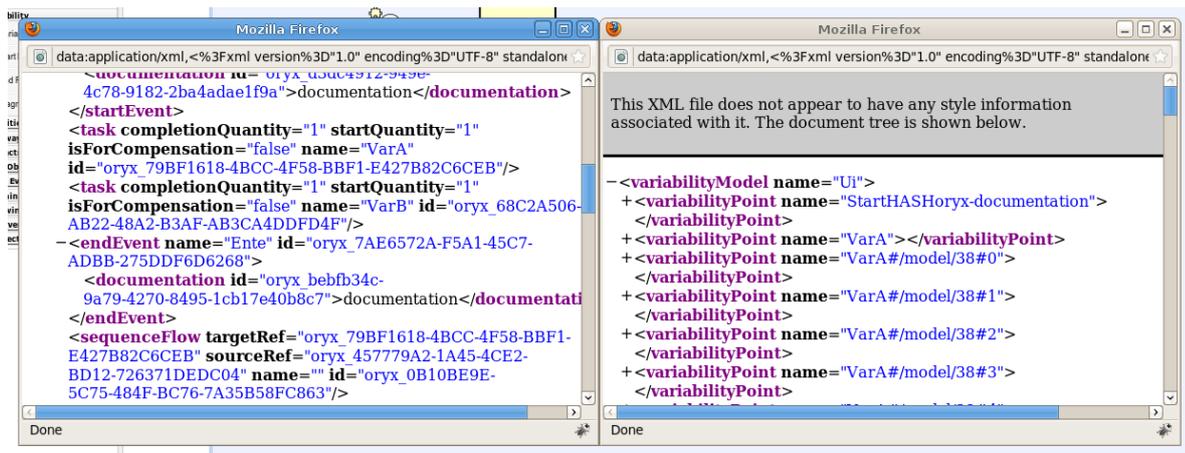


Abbildung A.26.: Exportierter Referenzprozess mit Variabilitätsbeschreibung

Der exportierte Referenzprozess wird wie in Abbildung A.26 in zwei separaten Fenstern angezeigt.

Literaturverzeichnis

- [ADWo8] A. Awad, G. Decker, M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In M. Dumas, M. Reichert, M. C. Shan, editors, *Proceedings of the 6th Int'l Conference on Business Process Management (BPM 2008)*, LNCS, pp. 326–341. Springer Verlag, Milano, Italy, 2008. (Zitiert auf Seite 88)
- [Allog9] T. Allweyer. *BPMN - Business Process Modeling Notation*. BoD, 2009. (Zitiert auf Seite 16)
- [Aueo8] K. Auer. *Generierung von WS-BPEL Prozessen aus Variabilitätsbeschreibungen*. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2008. URL http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2704&engl=0. (Zitiert auf den Seiten 15, 22, 24, 25 und 53)
- [Awa07] A. Awad. BPMN-Q: A Language to Query Business Processes. In M. Reichert, S. Strecker, K. Turowski, editors, *EMISA*, volume P-119 of *LNI*, pp. 115–128. GI, 2007. (Zitiert auf den Seiten 87 und 88)
- [Bar] J. Barrez. *jBPM goes BPMN!* <http://www.jorambarrez.be/blog/2009/12/04/jbpm-goes-bpmn/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [BGL⁺07] J. Bayer, C. Giese, T. Lehner, A. Ocampo, H. Overdick, F. Puhmann, A. Schnieders, J. Weiland, A. Werner, M. Weske. PESOA Abschlussbericht. PESOA-Report 29/2007, DaimlerChrysler Research and Technology, Hasso-Plattner-Institut, 2007. (Zitiert auf den Seiten 34 und 36)
- [CLRS04] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Algorithmen - eine Einführung*. Oldenbourger Wissenschaftsverlag, 2004. (Zitiert auf Seite 57)
- [Dino8] X. Ding. *Variability Points in WS-BPEL Prozessen*. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2008. URL http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2689&mod=0&engl=0&inst=IAAS. (Zitiert auf Seite 53)

- [GBDo6] L. Garca-Banuelos, M. Dumas. Towards an Open and Extensible Business Process Simulation Engine. In *In Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and CPN Tools (CPN '09)*. 2006. (Zitiert auf Seite 86)
- [GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns: Elements of reusable object-oriented Software*. Addison-Wesley Professional, 1995. (Zitiert auf den Seiten 63 und 69)
- [Gra] J. Graham. *Introduction to using XPath in JavaScript*. https://developer.mozilla.org/en/introduction_to_using_xpath_in_javascript. Zuletzt abgerufen am 29.10.2010. (Zitiert auf den Seiten 56 und 69)
- [Gui] A. Guizar. *Open Letter to the jBPM Community*. <http://enterprisebpm.blogspot.com/2010/03/open-letter-to-jbpm-community.html>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [HBRo8a] A. Hallerbach, T. Bauer, M. Reichert. Issues in Modeling Process Variants with Provop. In D. Ardagna, M. Mecella, J. Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pp. 56–67. Springer, 2008. (Zitiert auf Seite 34)
- [HBRo8b] A. Hallerbach, T. Bauer, M. Reichert. Modellierung und Darstellung von Prozessvarianten in Provop. In T. Kühne, W. Reisig, F. Steimann, editors, *Modellierung*, volume 127 of *LNI*, pp. 41–56. GI, 2008. (Zitiert auf den Seiten 31, 32, 34, 39 und 46)
- [HBRo9] A. Hallerbach, T. Bauer, M. Reichert. Guaranteeing Soundness of Configurable Process Variants in Provop. *E-Commerce Technology, IEEE International Conference on*, 0:98–105, 2009. (Zitiert auf den Seiten 34 und 39)
- [Hö] U. Hölzle. *Update on Google Wave*. <http://googleblog.blogspot.com/2010/08/update-on-google-wave.html>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [JvNo6] M. H. Jansen-vullers, M. Netjes. Business Process Simulation – A Tool Survey. In *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*. 2006. (Zitiert auf Seite 86)
- [Kun09] M. Kunze. *How to create a Stencil Set*. <http://code.google.com/p/oryx-editor/wiki/HowToCreateStencilSet>, 2009. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 28)
- [Mau09] C. Mauroner. *Ausführung von variantenbehafteten Workflow-Modellen in Abhängigkeit vom Prozesskontext*. Diplomarbeit, Universität Ulm, Germany, 2009. URL <http://dbis.eprints.uni-ulm.de/641/>. (Zitiert auf Seite 33)

- [Mieo8] R. Mietzner. Using Variability Descriptors to describe customizable SaaS Application Templates. Technical report, Universität Stuttgart; Fakultät Informatik, Elektrotechnik und Informationstechnik. Institut für Architektur von Anwendungssystemen, 2008. URL <http://elib.uni-stuttgart.de/opus/volltexte/2008/3425/>. (Zitiert auf den Seiten 14, 21, 22, 23, 24 und 25)
- [Mie10] R. Mietzner. *A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing*. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2010. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIS-2010-02&engl=0. (Zitiert auf Seite 21)
- [MLo8] R. Mietzner, F. Leymann. Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pp. 359–366. IEEE Computer Society, Washington, DC, USA, 2008. (Zitiert auf den Seiten 21 und 25)
- [Noro8] L. Northrop. *Software Product Line Essentials*. <http://www.sei.cmu.edu/library/assets/spl-essentials.pdf>, 2008. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 34)
- [NS] C. Neijenhuis, K. Schlichting. *Oryx Stepthrough-Plugin*. <http://code.google.com/p/oryx-editor/source/browse/trunk/editor/client/scripts/Plugins/stepThroughPlugin.js>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 86)
- [OADAo6] C. Ouyang, W. M. P. van der Aalst, M. Dumas, Arthur. From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way. Technical Report BPM-06-27, BPM Center, 2006. URL <http://www.bpmcenter.org>. (Zitiert auf Seite 16)
- [OMG09a] OMG. *BPMN 1.2 Spezifikation*. <http://www.omg.org/spec/BPMN/1.2/>, 2009. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 16)
- [OMG09b] OMG. *BPMN 2.0 Spezifikation*. <http://www.omg.org/spec/BPMN/2.0/>, 2009. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 16)
- [PBL05] K. Pohl, G. Böckle, F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. (Zitiert auf den Seiten 13 und 14)
- [Pet07] N. Peters. *Oryx Stencil Set Specification (Bachelor's thesis)*. <http://oryx-editor.googlecode.com/files/OryxSSS.pdf>, 2007. (Zitiert auf den Seiten 26 und 27)

- [Pol07] D. Polak. Oryx BPMN Stencil Set Implementation (Bachelor's thesis). <http://oryx-editor.googlecode.com/files/Oryxplementation.pdf>, 2007. (Zitiert auf Seite 28)
- [PP05] D. Pilone, N. Pitman. *UML 2.0 in a Nutshell (In a Nutshell (O'Reilly))*. O'Reilly Media, Inc., 2005. (Zitiert auf Seite 35)
- [PSWW05] F. Puhmann, A. Schnieders, J. Weiland, M. Weske. Variability Mechanisms for Process Models. PESOA-Report 17/2005, DaimlerChrysler Research and Technology, Hasso-Plattner-Institut, 2005. (Zitiert auf den Seiten 35 und 36)
- [RRHB09] M. Reichert, S. Rechtenbach, A. Hallerbach, T. Bauer. Extending a Business Process Modeling Tool with Process Configuration Facilities: The Provop Demonstrator. In *CEUR proceedings of the BPM'09 Demonstration Track, Business Process Management Conference 2009 (BPM'09), September 2009, Ulm, Germany*, volume 489. 2009. (Zitiert auf den Seiten 34 und 39)
- [SALM09] D. Schleicher, T. Anstett, F. Leymann, R. Mietzner. Maintaining Compliance in Customizable Process Models. In *OTM '09: Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems*, pp. 60–75. Springer-Verlag, Berlin, Heidelberg, 2009. (Zitiert auf den Seiten 88, 89 und 90)
- [Scha] V. A. Schmidt. *Das Internet der Dienste*. <http://de.sap.info/das-internet-der-dienste/12020>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 9)
- [Schb] T. Schreiter. *BPMN 2.0 Choreographiemodellierung*. http://www.bpmb.de/images/HU_Berlin_BPMN2_5_Schreiter_Choreographie.pdf. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 16)
- [Sil09a] B. Silver. *BPMN Method and Style: A levels-based Methodology for BPM Process Modeling and Improvement using BPMN 2.0*. Cody-Cassidy Press, 2009. (Zitiert auf den Seiten 19 und 42)
- [Sil09b] B. Silver. *BPMN 2.0 Status Update*. <http://www.brsilver.com/2009/07/06/bpmn-20-status-update-2/>, 2009. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 19)
- [Tsc07] W. Tscheschner. Oryx Dokumentation (Bachelor's thesis). <http://oryx-editor.googlecode.com/files/Bachelorner.pdf>, 2007. (Zitiert auf den Seiten 28, 59 und 60)
- [WB10] S. Wagner-Boysen. *How to develop an Editor Plugin*. <http://code.google.com/p/oryx-editor/wiki/HowToDevelopAnEditorPlugin>, 2010. Zuletzt abgerufen am 29.10.2010. (Zitiert auf den Seiten 26 und 29)

- [Wero8] A. Werner. *Mass Customization von Geschäftsprozessen – ein Rahmenkonzept zur familienbasierten Geschäftsprozessgestaltung*. Dissertation, Universität Leipzig, 2008. (Zitiert auf Seite 37)
- [Wil] J. Willms. *Evolutionäre Algorithmen: Chancen für die praxisorientierte Optimierung*. <http://www.fh-meschede.de/public/willms/ea/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 84)
- [WK] M. Weske, M. Kunze. *Prozessmodellierungsplattform Sommersemester 2010 Themen*. <http://bpt.hpi.uni-potsdam.de/pub/Public/StudentsCorner/pmp2010.themen.pdf>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 86)
- [WM09] A. Werner, H. Müller. Prozesskonfigurator – Softwaregestützte Geschäftsprozessberatung. *ERP Management*, 5:25–28, 2009. (Zitiert auf den Seiten 37 und 38)
- [WWWa] *The Oryx Editor - Homepage*. <http://bpt.hpi.uni-potsdam.de/Oryx/WebHome>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 25)
- [WWWb] *BPMN 2.0 Poster*. <http://www.bpmn.de/index.php/BPMNPoster>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf den Seiten 19 und 41)
- [WWWc] *JavaScript Object Notation*. <http://www.json.org>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf den Seiten 27 und 51)
- [WWWd] *Activiti Components*. <http://www.activiti.org/components.html>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [WWWe] *jBPM Homepage*. <http://www.jboss.org/jbpm>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [WWWf] *Oryx Editor on Google Code*. <http://code.google.com/p/oryx-editor>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 25)
- [WWWg] *Java Architecture for XML Binding (JAXB)*. <http://www.oracle.com/technetwork/articles/javase/index-140168.html>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 74)
- [WWWh] *ProcessWave.org*. <http://www.processwave.org/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 82)
- [WWWi] *ProcessWave auf Google Code*. <http://code.google.com/p/processwave/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 82)
- [WWWj] *Google Wave*. <http://wave.google.com>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 82)
- [WWWk] *Current Implementations of BPMN*. http://www.bpmn.org/BPMN_Supporters.htm. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 19)

- [WWWI] *BPMN-Q*. <http://bpmnq.sourceforge.net/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 87)
- [WWWm] *Web Services Business Process Execution Language Version 2.0*. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 88)
- [WWWn] *Signavio Process Editor – Software as a Service*. <http://www.signavio.com/en/products/process-editor-as-a-service.html>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 25)
- [WWWo] *Activiti BPM Suite*. <http://www.activiti.org/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [WWWp] *Activiti User Guide*. <http://activiti.org/userguide/index.html>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 83)
- [WWWq] *Home of Fundamental Modeling Concepts*. <http://www.fmc-modeling.org/>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 59)
- [WWWr] *openXchange - Fraunhofer-Institut für Arbeitswissenschaft und Organisation*. <http://www.e-business.iao.fraunhofer.de/projekte/beschreibung/openxchange.jsp>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 13)
- [WWWs] *BPMN Informaton Home - Object Management Group*. <http://www.bpmn.org>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf den Seiten 15 und 19)
- [WWWt] *Process Family Architect Demonstrationsfilm*. http://web.archive.org/web/20070720020626/http://www.pesoa.org/pages/Publications/DemosPresentations/ProcessFamilyArchitect_Demo.avi. Archivierete Webseite, zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 39)
- [WWWu] *Prozesskonfigurator – Download*. <http://www.prozesskonfigurator.de/download>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 39)
- [WWWv] *XML Path Language*. <http://www.w3.org/TR/xpath>. Zuletzt abgerufen am 29.10.2010. (Zitiert auf Seite 22)

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Falko Michael Kötter)