Institut für Softwaretechnik

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Ein Referenzmetamodell für Geschäftsprozessmodellierungssprachen

Studienarbeit
im Studiengang Informatik

vorgelegt von:
Grégory Catellani
gcatellani@uni-koblenz.de
Version 0.99

Betreuer: Prof. Dr. Andreas Winter, Carl von Ossietzky Universität Oldenburg, Abteilung Software-Engineering
Tassilo Horn, Universität Koblenz-Landau, Institut für Softwaretechnik
Judith Haas, Universität Koblenz-Landau, Institut für Softwaretechnik

Koblenz, im März 2010

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, den . . . .

Inhaltsverzeichnis

I. Einführung in die Thematik **1**

1. Einleitung in die modellbasierte Migration zu service-orientierten Architekturen **2**
1.1. Problemstellung und Lösungsidee . . . 3

II. Beschreibung ausgesuchter Geschäftsprozessmodellierungssprachen **5**

2. Einleitung **6**

3. *Unified Modeling Language* (*UML*) Aktivitätsdiagramme **8**
3.1. Einführung in die *UML* . . . 8
3.2. Notationsübersicht . . . 9
3.3. Notationselemente . . . 11
3.3.1. *Activities – Activity Nodes* . . . 11
3.3.1.1. *Action* . . . 12
3.3.1.2. *Activity* . . . 13
3.3.1.3. *Call Action* . . . 14
3.3.2. *Flow Controls* . . . 14
3.3.2.1. *Activity Edges* . . . 15
3.3.2.2. *Events* . . . 16
3.3.2.3. *Control Nodes* . . . 18
3.3.3. *Resources – Objects* . . . 20
3.3.4. *Participant Partitions – Activity Partitions* . . . 21
3.3.5. *Annotations – Comments* . . . 22

4. *Business Process Modeling Notation* (*BPMN*) **24**
4.1. Einführung in die *BPMN* . . . 24
4.2. Notationsübersicht . . . 25
4.3. Notationselemente . . . 27
4.3.1. *Activities* . . . 27
4.3.1.1. *Task* . . . 27
4.3.1.2. *Sub-Process* . . . 28
4.3.1.3. *Reusable Sub-Process – Call Task* . . . 29
4.3.2. *Flow Controls* . . . 30
4.3.2.1. *Flows* . . . 30
4.3.2.2. *Events* . . . 31

4.3.2.3. *Gateways* . . . 34
4.3.3. *Resources – Data Objects* . . . 36
4.3.4. *Participant Partitions – Swimlanes* . . . 37
4.3.5. *Annotations – Text Annotations* . . . 39
4.4. Fazit . . . 40

5. *Ereignisgesteuerte Prozessketten* (EPK) 41
5.1. Einführung in die EPK . . . 41
5.2. Notationsübersicht . . . 42
5.3. Notationselemente . . . 45
5.3.1. *Activities – Functions* . . . 45
5.3.1.1. *Atomic Function* . . . 45
5.3.1.2. *Complex Function* . . . 46
5.3.1.3. *Call Function* . . . 47
5.3.2. *Flow Controls* . . . 47
5.3.2.1. *Control Flow* . . . 48
5.3.2.2. *Events* . . . 48
5.3.2.3. *Connectors* . . . 50
5.3.3. *Resources – Information Objects* . . . 52
5.3.4. *Participant Partitions – Organizational Units* . . . 53
5.3.5. *Annotations – Descriptions* . . . 54
5.4. Fazit . . . 54

III. Erstellung einer Referenz für UML Aktivitätsdiagramme, BPMN und EPK 55

6. Einleitung 56

7. Vergleich der Sprachkonzepte und Notationselemente 57
7.1. *Activities* . . . 57
7.1.1. *Atomic Process* . . . 58
7.1.2. Sub-Process . . . 58
7.1.3. Call Process . . . 59
7.2. Flow Controls . . . 59
7.2.1. Events . . . 60
7.2.1.1. Start Event . . . 63
7.2.1.2. Flow Final Event . . . 64
7.2.1.3. Process Final Event . . . 65
7.2.1.4. Signal Throwing Event . . . 66
7.2.1.5. Signal Catching Event . . . 67
7.2.2. Flow . . . 68
7.2.2.1. Resource Flow . . . 78
7.2.3. Connectors . . . 82
7.2.3.1. Exclusive Branch Connector . . . 83
7.2.3.2. Parallel Branch Connector . . . 83

7.2.3.3. Exclusive Merge Connector . . . 84
7.2.3.4. Parallel Merge Connector . . . 84
7.3. Resources . . . 84
7.4. Participant Partitions . . . 88
7.5. Annotations . . . 90
7.6. Vergleichstabelle . . . 92

8. Erhebung und Beschreibung von Metamodellen 93
8.1. Übersicht . . . 93
8.2. *Activities* . . . 96
8.3. *Flow Controls* . . . 98
8.3.1. *Events* . . . 99
8.3.2. *Flows* . . . 103
8.3.3. *Connectors* . . . 105
8.4. *Resources* . . . 107
8.5. *Participant Partitions* und Verschachtelungshierarchie . . . 109

IV. Exemplarische Realisierung eines Modell-zu-Referenzmodell Konverters für die *BPMN*-Notation 113

9. Einleitung 114

10. Konvertierung von graphischen *BizAgi*-*BPMN*-Modellen in *BPMN*-Metamodell-konforme Modelle 115
10.1. *Vergleich der Notationselemente aus BizAgi* und dem *BPMN*-Metamodell . . . 115
10.2. Funktionsweise des Konverters für die Überführung von *BizAgi*-Modellen in *BPMN*-Metamodell-konforme Modelle . . . 118

11. Transformation *BPMN*-Metamodell-konformer Modelle in Referenzmetamodell-konforme Modelle 119
11.1. Transformationsregeln . . . 119
11.1.1. Oberklassen und *Annotation* . . . 119
11.1.2. *Activities* . . . 120
11.1.3. *Flow Controls* . . . 121
11.1.3.1. *Events* . . . 121
11.1.3.2. *Flows* . . . 122
11.1.3.3. *Connectors* . . . 122
11.1.4. *Resources* . . . 123
11.1.5. *Participant Partitions* . . . 123
11.2. Anwendung der Transformationsregeln anhand eines modell-zu-referenzmodell Konverters . . . 124

12. Validierung des Ansatzes anhand von Beispielen 125
12.1. Beispiel: Multiple ein- & ausgehende Ausführungsflüsse . . . 125
12.2. Beispiel: Transport mehrerer Datensätze . . . 129

12.3. Beispiel: Einsatz von *BPMN Signal & Resource Events* 133
12.4. Beispiel: Einsatz von *BPMN Pools* 137
12.5. Beispiel: Geschäftsprozess *sellTrainticket* 146

V. Fazit **151**

13. Zusammenfassung **152**

VI. Appendix **153**

Appendix A – *Compact Disc*-Inhalt **154**

Textkonventionen

Die nachstehenden Textvorlagen werden im Dokument genutzt um Quellcode und Syntaxelemente von normaler deutschen Sprache zu trennen. Die Konventionen werden allerdings nicht in Tabellen oder Überschriften verwendet, wo eine Unterscheidung nicht nötig ist.

Palatino Linotype – 11pt.: Normaler Text

Palatino Linotype – 11pt.: Hervorzuhende Wörter, Notationselemente, Klassennamen, Attribute

Palatino Linotype – 11pt.: Fremdwörter sowie Namen von Projekten, Notationen, Programmiersprachen, Computerprogramme, Dateiformate, Institutionen oder Dokumenten wie Spezifikationen und Publikationen

PALATINO LINOTYPE – 11PT.: Nachnamen von Fachleuten

`Courier New – 11pt.`: Quellcode

`Courier New – 11pt.`: Klassennamen, Attributnamen, Assoziationsnamen

Teil I.

Einführung in die Thematik

1. Einleitung in die modellbasierte Migration zu service-orientierten Architekturen

Große kommerzielle Softwaresysteme aus dem Industrie- und Dienstleistungssektor basieren heute nicht selten noch auf Technologien aus den 70er und 80er Jahren. Diese softwaretechnischen Urgesteine – man spricht in diesem Zusammenhang auch von ***Legacy*-Systemen** – bergen aber **immer noch großes, wirtschaftliches Potential** und enthalten nicht selten das gesamte, unternehmensspezifische Wissen über das Softwaresystem sowie die Geschäftslogik.

Durch den stetigen Wandel in der Softwarelandschaft wird der **Abstand zwischen alten und neuen Technologien immer größer**. Das hat negative Auswirkungen auf die Betreiber solch alter Systeme, wie u.a. steigende Fixkosten für die Wartung, mangelnde Flexibilität, zurückgehende Programmierkenntnisse und die erschwerte Integration neuer Technologien. Um effizienzsteigernd zu arbeiten und konkurrenzfähig zu bleiben, müssen die Unternehmen ihre altgedienten **Systeme früher oder später ersetzen**. Nicht selten schrecken Unternehmer aber, auf Grund hoher **Kosten** und einem hohen **Aufwand**, die die Umstellung auf neue Technologieansätze, **durch eine komplette Neuprogrammierung** der Software sowie der damit oft einhergehenden Hardware-Neubeschaffung mit sich bringt, zurück (vgl. [SOA09]).

Einen alternativen Weg zur Neuprogrammierung geht das, vom *Bundesministerium für Bildung und Forschung* geförderte und vom *Deutschen Zentrum für Luft- und Raumfahrt* koordinierte, ***SOAMIG*-Projekt**[1]. Der Technologiewechsel wird hier durch den Ansatz der **modell-basierten Software-Migration** durchgeführt, das heißt durch die automatisierte Überführung der *Legacy*-Systeme in eine neue Umgebung mit Erhalt der Funktionalität . Die alten Anwendungen werden nach dem Prinzip des *modeldriven software developement* durch Modelltransformationen in aktuelle, service-orientierte Architekturen (*service oriented architecture, SOA*) überführt (vgl. [SOA09]).

Die Funktionalität eines Systems wird durch sein statisches und dynamisches Verhalten auf Datenmodell-, Code- und Architekturebene definiert und orientiert sich an den verfolgten Geschäftszielen und Anforderungen für/ an dieses Produkt. Einer modell-basierten Migration auf eine neue Architektur geht daher die **Analyse des *Legacy*-Systems** und die **Bestimmung von Softwareartefakten** auf verschiedenen Abstraktionsebenen durch *reverse engineering* Techniken voran, welche in Modelle aufgefasst werden (vgl. [JZ07]). Code-Modelle beschreiben auf der untersten Ebene, die internen, logischen Strukturen eines Systems. Architekturmodelle, u.a. durch *UML* Kompositionsstrukturdiagramme und *UML* Paketdiagramme repräsentiert, beschreiben die Komponenten eines Systems und ihre Beziehungen zueinander. Die **betrieblichen Abläufe** eines Systems und die Anforderungen an dieses System lassen sich mit **Geschäftsprozessen** modellieren, eine „logisch zusammengehörige Menge von Aktivitäten, welche einem bestimmten Geschäftszweck

[1] `http://www.soamig.de/`

dienen "[Mie02] und können aus Softwaresystemsischt als ein „mit Software durchgeführter Arbeitsablauf, der aber manuelle und organisatorische Anteile besitzen kann"[ebd.] aufgefasst werden. Geschäftsprozesse lassen sich mit Geschäftsprozessmodellierungssprachen/ -notationen wie u.a. den Aktivätsdiagrammen der *Unified Modelling Language* (*UML*), der *Business Process Modelling Notation* (*BPMN*) oder der *ereignis-gesteuerten Prozessketten* (*EPK*) beschreiben. Die bei der Analyse des Altsystems erhobenen Geschäftsprozesse werden bei der Migration auf Prozesse (***services***) der neuen service-orientierten Architektur abgebildet.

Der nächste Vorbereitungsschritt in einer modell-basierten Software-Migration besteht darin, die auf Code-, Architektur- und konzeptueller Ebene **erhobenen Artefakte** bzw. deren Komponenten miteinander zu **verknüpfen**. Zu diesem Zweck, sowie einer einfachen Weiterverarbeitung/ Anpassung der Modelle, sollen diese im Rahmen des *SOAMIG*-Projekts mit dem, am *Institut für Softwaretechnik der Universität Koblenz-Landau* entwickelten, Graphenlabor ***JGraLab***[2] eingelesen und bearbeitet werden. Hierfür werden die Modelle in das *TGraph*-Format auf Basis von typisierten, attributierten, gerichteten und angeordneten Graphen überführt, wo sie durch Graphschemas auf Metamodellebene syntaktisch beschrieben werden – in Analogie an Datenbankschemas aus der relationalen Datenbankprogrammierung. Aus diesen Graphschemas erzeugt *JGraLab* Klassen und Interfaces für eine objektorientierte Zugriffsschicht auf die Graphen.

Das **zusammengeführte Gesamtmodell der Legacy-Anwendung bildet die Basis für die Migration**, bei der gezielte Programmanpassungen durch Modelltransformationen durchgeführt werden.

1.1. Problemstellung und Lösungsidee

Im Vordergrund dieser Arbeit stehen die **Geschäftsprozessmodelle** und deren **Überführung** in das für eine Weiterverarbeitung in *JGraLab* benötigte *TGraph*-Format. Für die Modellierung der Geschäftsprozesse im *SOAMIG*-Projekt fiel die Auswahl, aus Gründen der Verbreitung und der Kompetenzen der *SOAMIG*-Partner, zugunsten der drei bereits erwähnten Modellierungssprachen *UML* Aktivitätsdiagramme, *BPMN* und *EPK* aus. Obwohl sich diese drei Notationen allesamt für die Erstellung von Geschäftsprozessmodellen bewährt haben und auch in Kombination in der Praxis eingesetzt werden, so differenzieren sie sich dennoch durch verschiedene Zielsetzungen bei ihrer Entwicklung. Die aufgrund der Zielsetzungen und der hohen Funktionsvielfalt resultierenden **Unterschiede und Inkompatibilitäten der Sprachen** untereinander bei der Modellierung von Geschäftsprozessen erschweren eine einheitliche Transformation nach *JGraLab* und die Behandlung der zu erhebenden Geschäftsprozessmodelle in *JGraLab*. Für jede Sprache müsste ein eigener Regelsatz und eine maßgeschneiderte Konvertierungsmöglichkeit festgelegt werden.

Um eine **sprach- und werkzeugunabhängige Geschäftsprozessmodellierung** zu ermöglichen werden in dieser Arbeit zuerst die im *SOAMIG*-Projekt **benötigten Sprachkonzepte und -elemente** für die Modellierung von Geschäftsprozessen der ***UML*** **Aktivitätsdiagramme**, der ***BPMN*** und der ***EPK*** vorgestellt (siehe Teil II). Im nächsten Schritt werden **notationsübergreifende Abbildungen der Srachkonzepte und -elemente** aufeinander beschrieben, Defizite und Lücken der Sprachen aufgedeckt und durch eigene Auslegungen der Notationen gefüllt, und schließlich **Metamodelle der drei Sprachen**, aufgrund der gewonnen Erkentnisse sowie ein **Referenzmetamodell**

[2] `http://userpages.uni-koblenz.de/~ist/JGraLab`

auf Basis dieser Metamodelle erstellt und dargestellt (siehe Teil III). Im letzten Teil dieser Arbeit wird die **exemplarische Realisierung** eines Modell-zu-Referenzmodell Konverters **auf Basis der BPMN** und des Modellierungswerkzeuges *BizAgi Process Modeler* der Firma *BizAgi Ltd*[3] vorgestellt, und der **Lösungsansatz** anhand des Beispiels eines Geschäftsprozesses **validiert** (siehe Teil IV). Die Konvertierer ermöglichen die Abbildung eines Geschäftsprozessmodelles auf ein semantisch-äquivalentes Referenzmetamodell-konformes Modell, auf dessen Basis weitere Arbeitsschritte vor der/ für die Migration, unabhängig von der ursprünglich eingesetzten Modellierungssprache, definiert werden können.

[3] `http://www.bizagi.com/`

Teil II.

Beschreibung ausgesuchter Geschäftsprozessmodellierungssprachen

2. Einleitung

Bevor eine Referenz für Geschäftsprozessmodellierungssprachen entwickelt werden kann, muss festgehalten werden, welche Konzepte und Sprachelemente die Referenz enthalten und abbilden können soll. Dieses Kapitel beschreibt daher die Hauptelemente, welche für die Geschäftsprozessmodellierung im Rahmen des *SOAMIG*-Projektes benötigt werden. Jeder Sprache wird im Folgenden ein Kapitel gewidmet. Da alle zu diesem Zeitpunkt erstellten Geschäftsprozessmodelle im *SOAMIG*-Projekt als ***UML*** **Aktivitätsdiagramme** vorliegen, **dient** diese Notation dieser Arbeit **als Referenz** für Notationskonzepte und Sprachelemente. *UML* Aktivitätsdiagramme werden daher als erstes behandelt (siehe Kapitel 3). Als nächstes wird die *BPMN* (siehe Kapitel 4) und schließlich die *EPK* (siehe Kapitel 5) beschrieben. Die folgenden Kapitel bestehen aus einer **kurzen Einführung in die Entwicklung** der jeweiligen Sprache, einem **kurzen Einführungsbeispiel** in Form eines, mit der jeweiligen Notation modellierten, Geschäftsprozesses, gefolgt von einer **Beschreibung der Sprachkonzepte und Notationselemente** für die Modellierung von Geschäftsprozessen.

Anzumerken ist, dass die **Struktur der Unterkapitel** zur Beschreibung der Sprachelemente sich **nicht an der Klassifizierung der Elemente aus den offiziellen Sprachspezifikationen orientiert, sondern auf den Sachverhalt von Geschäftsprozessen** ausgerichtet ist. Die Strukturierung orientiert sich an den Grundkonzepten, welche sich in einem Geschäftsprozess identifizieren lassen und sich auch in den drei Modellierungssprachen wiederfinden. So werden die Tätigkeiten, die zur Erreichung des Ziels eines Geschäftsprozesses führen, unter dem Begriff ***Activities*** zusammengefasst. Der Prozessablauf und dessen Steuerung findet sich unter der Bezeichnung ***Flow Controls*** wieder. Die Verantworungsbereiche der am Geschäftsprozess teilnehmenden Akteure (***Participants***) werden unter dem Schlüsselwort ***Participant Paritions*** dargestellt, und die während der Ausführung anfallende Daten unter dem Punkt ***Resources***. Weiterführende textuelle und formale Erklärungen sowie Anmerkungen zum Ablauf des Geschäftsprozesses oder einzelner Schritte stehen unter dem Punkt ***Annotations***.

Ausgehend von dieser Klassifizierung lassen sich aus den, im *SOAMIG*-Projekt erhobenen, Geschäftsprozessen die folgenden benötigten Sprachkonzepte und Notationselemente zur Modellierung von Geschäftsprozessen identifizieren:

- ***Activities*** sind atomare und zusammengesetzte Tätigkeiten.
- ***Flow Control*s** beschreiben die Steuerung des Ausführungsflusses und den Austausch von Datensätzen zwischen ***Activities***, die Funktionalität zum Auslösen und Auswerten von Ereignissen in einem Geschäftsprozess, parallele und alternative Abläufe, sowie Ausführungsstart und -ende von Geschäftsprozessen.
- ***Resource*s** stellen die dem Geschäftsprozess zugeführten und abgeführten sowie, bei der Ausführung des Geschäftsprozesses, aufkommenden und zwischen ***Activities*** und ***Participants*** ausgetauschten Daten dar.

- ***Participant Paritions*** beschreiben die Verantwortungsbereiche der an der Ausführung eines Geschäftsprozesses beteiligten Akteure (***Participants***) , d.h. ***Activities***, die von diesem Akteur abgearbeitet werden. Bei einem Akteur handelt es sich um einen Typ oder eine Rolle die u.a. eine Firma, ein System, eine Personengruppe oder ein einzelnes Individuum bei der Ausführung des Prozesses einnehmen kann (vgl. [Kec06]).
- ***Annotation*****s** bieten dem Modellierer die Möglichkeit weitere Anmerkungen zum gesamten Geschäftsprozess oder einzelnen Schritten des Geschäftsprozesses anzugeben.

Falls eine Sprache ein namentliches Gegenstück zu einem Kategorienamen aufweist, wird in der Unterkapitelüberschrift darauf hingewiesen und in den folgenden Beschreibungen der Begriff aus der Notation verwendet.

3. *Unified Modeling Language* (*UML*) Aktivitätsdiagramme

3.1. Einführung in die *UML*

In den 90er Jahren gab es im Bereich der objektorientierten Modellierung eine Vielzahl verschiedener Modellierungsansätze. Daher stieß die Firma Rational *Software*, welche später in *IBM* aufging, die Entwicklung einer neuen Modellierungssprache an, welche die unterschiedlichen objektorientierten Paradigmen vereinen sollte. Unter Federführung von Grady BOOCH, James RUMBAUGH und Ivar JACOBSON sowie der Unterstützung unterschiedlicher Interessenten aus verschiedenen Industriezweigen wurde 1996 der Grundstein für die *Unified Modeling Language* (*UML*) gelegt und 2007 der *Object Management Group*[1] (*OMG*) als Version 1.0 zur Standardisierung vorgelegt. In den Folgejahren wuchs und reifte die Sprache weiter, wurde aber aufgrund ihrer ansteigenden Komplexität und der Größe ihrer Spezifikation kritisiert (vgl. [Mey97], [Jac06]). Daraufhin begann die *OMG* 1999 die Entwicklung einer sanierten und überarbeiteten Version, welche 2005 fertiggestellt und als Version 2.0 angenommen wurde. Aktuell ist Version 2.2, welche sich nicht wesentlich von den Vorgängerversionen seit 2005 unterscheidet.

Die *UML* bietet eine Vielzahl von Sprachkonzepten und Diagrammarten für die Anwendung im Bereich der Softwareentwicklung aber auch im Bereich der betrieblichen Anwendungssoftware. Ihre Beschreibung verteilt sich auf unterschiedliche Spezifikationsmanifeste. Während die *UML Infrastructure* das Kernmodell der Sprache beschreibt, definiert die darauf aufgebaute *UML Superstructure* die Notation und Semantik der Diagramme und Elemente. Die Sprachkonzepte sind in der *UML Superstructure* auf zwei große Bereiche aufgeteilt. Während Strutkturdiagramme statische, zeitunabhängige Elemente eines Systems darstellen, modellieren Verhaltensdiagramme die dynamischen Aspekte und das Verhalten eines Systems und seiner Komponenten (vgl. [Kec06]). Diese Bereiche werden weiterhin unterteilt in Spracheinheiten, welche bestimmte Konzepte beschreiben die in Diagrammen verwendet werden. Für die Modellierung von Geschäftsprozessen eignen sich vor allem die *Action*- und *Activi*ty-Konzepte, welche u.a. in Anwendungsfall- und Aktivitätsdiagrammen Anwendung finden.

Mit Anwendungsfalldiagrammen (*Use Case Diagrams*) wird die Funktionalität eines Systems auf einem hohen Abstraktionsniveau modelliert. Anwendungsfalldiagramme beschreiben welche Funktionalität das System oder, übertragen auf die Geschäftswelt, welche Dienstleistungen eine Firma anbietet. Innere Abläufe wie sie in Geschäftsprozessen modelliert werden, werden bei dieser Diagrammart nicht umgesetzt.

Für die Modellierung von Geschäftsprozessen sind nach der *UML Superstructure* die Aktivitätsdiagramme (*Activity Diagrams*) geeigneter, obwohl dies nicht ihr primärer Einsatzbereich ist (vgl.

[1] `http://www.omg.org/`

[OMG09b]). Aktivitätsdiagramme bieten viele Möglichkeiten das Verhalten eines Systems, bzw. den inneren Ablauf eines Geschäftsprozesses darzustellen.

Für die Modellierung von Geschäftsprozessen in *SOAMIG* mit der *UML* im Allgemeinen und den Aktivitätsdiagrammen im Speziellen, sprechen vor allem der große Funktionsumfang der Sprache, welche alle gebräuchlichen Konzepte für die Geschäftsprozessmodellierung behandelt. Außerdem bietet die *UML* eine präzise und umfassende Spezifikation. Als Wehrmutstropfen stellt sich die Größe und Komplexität der Sprache und Spezikation heraus, was wohl einer der Gründe ist warum sich die Notation in der Praxis nicht für die Geschäftsprozessmodellierung durchsetzen konnte (vgl. [All08]).

3.2. Notationsübersicht

Zur Einführung in die *UML* Aktivitätsdiagramme wird in Abbildung 3.1 die Modellierung eines Geschäftsprozesses aus dem *SOAMIG*-Projekt dargestellt. Der Geschäftsprozess wurde durch den Autor abgewandelt um weitere Notationselemente im Modell vorzustellen.

Der modellierte Geschäftsprozess stellt einen vereinfachten Arbeitsablauf bei der Bearbeitung einer Fahrkartenbestellung durch einen Angestellten eines Reisebüros dar. Der Bestellungsablauf *sellTrainticket* wird als ***Activity*** (siehe Unterkapitel 3.3.1.2), d.h. als komplexer Zusammenhang mehrerer Einzelabläufe oder, da hiermit der gesamte Geschäftsprozess abgebildet wird, als Aktivitätsdiagramm (***Activity Diagram***, siehe Unterkapitel 3.3.1.2) bezeichnet. Ein Geschäftsprozessakteur wie ein Reisebüroangestellter (*Employee*) wird in *UML* Aktivitätsdiagrammen als ***Activity Partition*** (siehe Unterkapitel 3.3.4) dargestellt, d.h. der für diesen Bereich eingeteilte Akteur ist für die Ausführung der darin enthaltenen Geschäftsvorgänge zuständig.

Der Startpunkt des Geschäftsprozesses ist ein ***Initial Node*** (siehe Unterkapitel 3.3.2.2), von hier aus läuft der Ausführungsfluss über die als ***Control Flow*** (siehe Unterkapitel 3.3.2.1) bezeichneten Kanten weiter. Bevor der Reisebüroangestellte jedoch eine Bestellung bearbeiten kann, muss er sich in das Farkartensystem einloggen. Dieser zusammengesetzte Arbeitsablauf ist als *logIntoSystem*-***Activity*** beschrieben. Zum Einloggen meldet sich der Angestellte gleichzeitig mit Passwort und Identifikationskarte an – *enterPassword* und *insertIdCard*. Hierbei handelt es sich um einfache, atomare Abläufe welche in *UML* Aktivitätsdiagrammen als ***Action*** (sieh Unterkapitel 3.3.1.1) bezeichnet werden. Die gleichzeitige Eingabe wird durch ein ***Fork Node*** (siehe Unterkapitel 3.3.2.3) hervorgerufen und durch ein ***Join Node*** (siehe Unterkapitel 3.3.2.3) synchronisiert. Stimmen die Sicherheitsmerkmale mit den abgespeicherten Werten im System überein, terminiert die *logIntoSystem*-***Activity*** erfolgreich an einem ***Activity Final Node*** (siehe Unterkapitel 3.3.2.2). Andernfalls ist die erneute Eingabe durch den Angestellten nötig. Ein ***Decision Node*** (siehe Unterkapitel 3.3.2.3) stellt anhand von Ausführungsbedingungen (***Guards***) sicher, dass nur ein zutreffender Ausführungszweig durchlaufen wird, ein ***Merge Node*** (siehe Unterkapitel 3.3.2.3) führt mehrere eingehende Ausführungszweige zusammen.

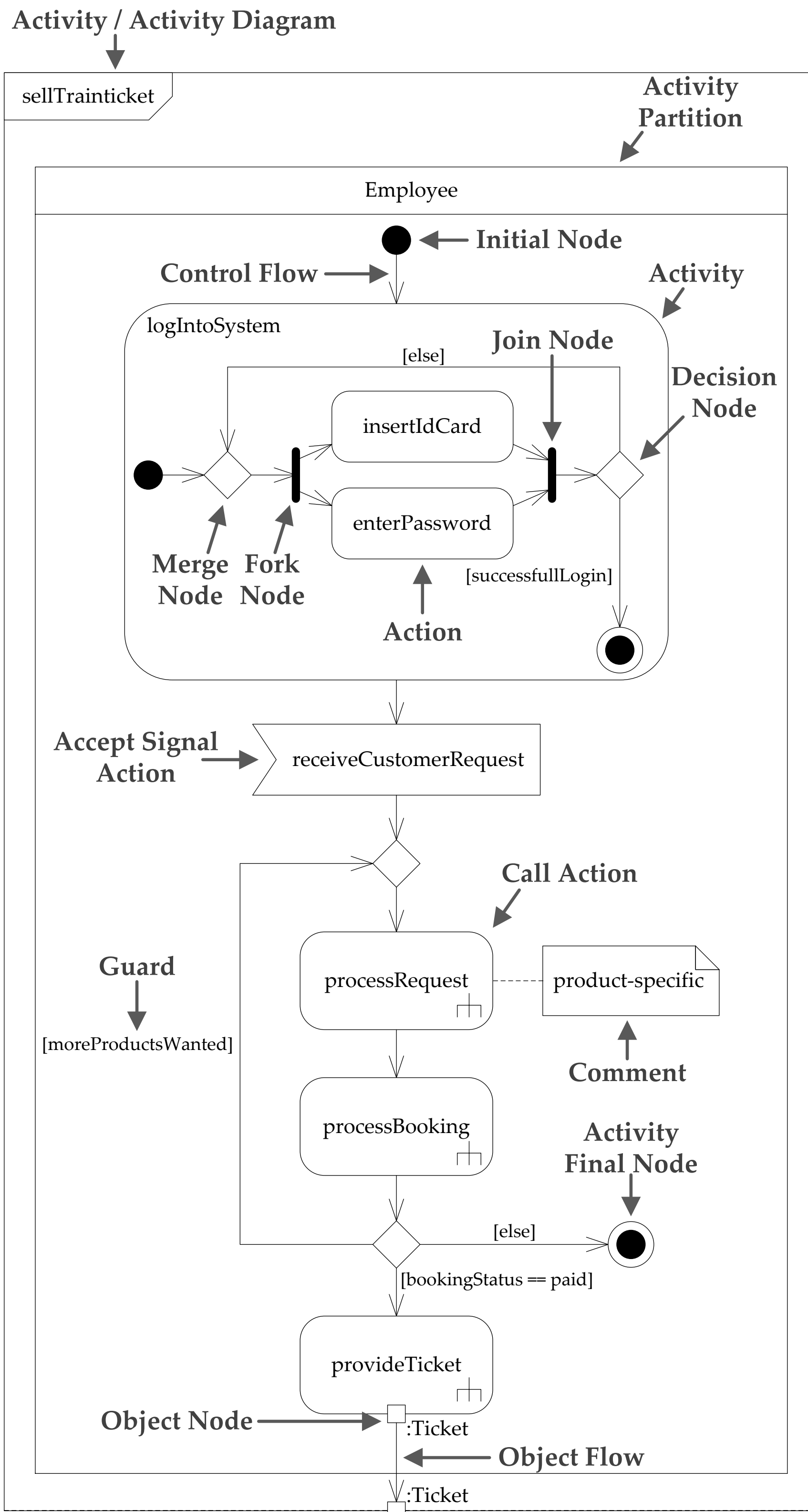


Abbildung 3.1.: Modellierung eines Geschäftsprozesses als *UML* Aktivitätsdiagramm

Der Angestellte ist zu diesem Zeitpunkt bereit die Anfrage eines Kunden zu bearbeiten. Tritt dieses Ereignis ein, wird das ***Accept Signal Event*** *receiveCustomerRequest* (siehe Unterkapitel 3.3.2.2) abgearbeitet und der Ausführungsfluss weitergeführt. Der Büroangestellte setzt Angaben und Wünsche des Kunden in einer zusammengesetzten, aber nicht abgebildeten ***Activity*** *processRequest* um, welche als gleichnamiger Geschäftsprozess in einem weiteren ***Activity Diagram*** modelliert sein kann. Beim Aufruf des an einer anderen Stelle festgelegten Verhaltens einer ***Activity***/ eines ***Activity Diagrams***, wird von einer ***Call Action*** (siehe Unterkapitel 3.3.1.3) gesprochen, da der Verhaltensaufruf wie eine ***Action*** atomar ist. Die in *processRequest* nötigen Bearbeitungsschritte sind vom gewählten Produkt abhängig, wie an dem angehefteten ***Comment*** (siehe Unterkapitel 3.3.5) abzulesen.

Nach der Bearbeitung der Kundenanfrage wird durch eine weitere ***Call Action*** die ***Activity*** *processBooking* aufgerufen, welche für die Buchung der Auswahl zuständig ist. Entscheidet sich der Kunde für weitere Produkte, wird am folgenden ***Decision Node*** der ***Control Flow*** mit dem ***Guard*** *moreProductsWanted* eingeschlagen und der Ausführungsfluss geht zurück zur ***Call Action*** *processRequest*. Entscheidet sich der Kunde die bereits gebuchten Produkte zu zahlen, wird dem Kunden im Zuge der ***Call Action*** *provideTicket* die nunmehr gekauften Fahrkarten übergeben. An dieser Stelle verlässt der Ausführungsfluss den modellierten Geschäftsprozess über einen spezielle ***Control Flow***, ein ***Object Flow*** (siehe Unterkapitel 3.3.2.1), welche den Ausführungsfluss mit den Fahrkarten als Datensatz vorantreibt, in *UML* ***Object Node*** (siehe Unterkapitel 3.3.3) genannt. Wird das Geschäftsprozessmodell in ein weiteres Modell eingebettet, könnte der Ausführungsfluss an dieser Stelle weiterlaufen. Verzichtet der Kunde auf die bereits gebuchten Produkte, endet die Transaktion zwischen Kunde und Reisebüroangestelltem und damit der gesamte Geschäftsprozess *sellTrainticket* an einem ***Activity Final Node***.

3.3. Notationselemente

Im folgenden werden die Sprachkonzepte und Notationselemente der *UML* Aktivitätsdiagramme beschrieben, die sich in die in der Einleitung zu diesem Kapitel (siehe Kapitel 2) identifizierten Elementgruppen einordnen lassen und für die Erhebung von Geschäftsprozessmodellen im *SOAMIG*-Projekt benötigt werden. Die Elemente sind den folgenden fünf Kategorien zugeordnet:

- ***Activities – Activity Nodes*** (siehe Unterkapitel 3.3.1),
- ***Flow Controls*** (siehe Unterkapitel 3.3.2),
- ***Resources – Object Nodes*** (siehe Unterkapitel 3.3.3),
- ***Participant Partitions – Activitiy Partitions*** (siehe Unterkapitel 3.3.4) und
- ***Annotations – Comments*** (siehe Unterkapitel 3.3.5).

3.3.1. *Activities – Activity Nodes*

Diese Kategorie beinhaltet Sprachelemente zur Beschreibung des Verhaltens eines *UML* Aktivitätsdiagrammes und bildet einfache und komplexe Tätigkeiten eines Geschäftsprozesses ab. Es wird

unterschieden zwischen:

- ***Actions*** (siehe Unterkapitel 3.3.1.1),
- ***Activities*** (siehe Unterkapitel 3.3.1.2) und
- ***Call Actions*** (siehe Unterkapitel 3.3.1.3).

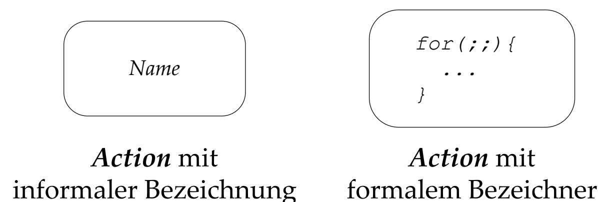
Im weiteren Verlauf dieser Arbeit wird für Notationselemente aus dieser Kategorie von ***Activity Nodes*** gesprochen.

3.3.1.1. *Action*

Unter dem Begriff ***Action*** versteht die *UML* eine atomare Tätigkeit in einem Aktivitätsdiagramm. Eine ***Action*** modelliert einfache, kurze Funktionalitäten, sowie umfangreiche, nicht atomare Tätigkeiten, die jedoch im Modell/ Geschäftsprozess als atomar angesehen und nicht weiter aufgeteilt werden (vgl. [Kec06]).

Eine ***Action*** wird über mindestens eine ein- und mindestens eine ausgehende ***Activity Edge*** (siehe Unterkapitel 3.3.2.1) in den Ausführungsfluss eingebunden, d.h. über gerichtete Kanten mit anderen Notationselementen verbunden. Die Ausführungsreihenfolge von ***Actions*** wird durch ***Flow Controls*** vorgegeben (siehe Unterkapitel 3.3.2). Über ***Object Nodes*** (siehe Unterkapitel 3.3.3) und ***Object Flows*** (siehe Unterkapitel 3.3.2.1) können einer ***Action*** Datensätze zugeführt werden, welche bei der Ausführung der ***Action*** von Bedeutung sein können. Genau so können zugeführte oder bei der Ausführung produzierte Daten über ***Object Nodes*** und ***Object Flows*** weitergegeben werden (siehe Unterkapitel 3.3.2.1).

Graphisch werden ***Actions*** als Rechtecke mit abgerundeten Ecken dargestellt. Der Name der ***Action*** – eine informale Bezeichnung der Tätigkeit die sie ausführt – wird an einer beliebigen Stelle in das Rechteck eingefügt (siehe Abb. 3.2). Die *UML*-Spezifikation sieht zudem die Möglichkeit vor, den Namen durch einen formalen Ausdruck zu ersetzen (siehe Abb. 3.2). Dieser kann in Pseudocode oder einer beliebigen Programmiersprache verfasst sein und legt das Verhalten der ***Action*** beliebig detailliert fest.



Action mit informaler Bezeichnung ***Action*** mit formalem Bezeichner

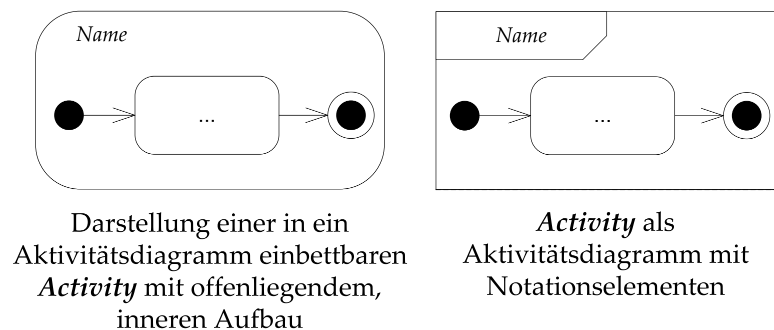
Abbildung 3.2.: Graphische Darstellung von ***Actions***

3.3.1.2. *Activity*

Activities bilden nicht-atomare Zusammenhänge und beschreiben ein komplexes Verhalten. Mit ***Activities***, bzw. mit dem einer ***Activity*** zugrunde liegenden *UML* Aktivitätsdiagramm lassen sich ganze Geschäftsprozesse darstellen. ***Activities*** setzen sich aus weiteren Notationselementen aus *UML* Aktivitätsdiagrammen zusammen und können weitere ***Activities*** enthalten. Allerdings kann eine ***Activity*** nicht in sich selbst vorkommen.

Eine ***Activity*** kann auf zwei Arten in ein Aktivitätsdiagramm eingebunden werden:

- Wenn der innere Aufbau einer ***Activity*** für das Verständnis eines Aktivitätsdiagrammes nicht notwendig ist, oder der Inhalt noch nicht modelliert wurde, kann das Verhalten einer ***Activity*** durch eine ***Call Action*** aufgerufen werden (siehe Unterkapitel 3.3.1.3). Der Inhalt der ***Activity*** wird nicht dargestellt.
- Steht der innere Aufbau einer ***Activity*** fest und wird dessen komplette Darstellung in einem Aktivitätsdiagramm benötigt, liegt der innere Aufbau dieser ***Activity*** offen, d.h. alle graphischen Elemente werden dargestellt. Die graphische Repräsentation einer solchen ***Activity*** unterscheidet sich nur leicht von der einer ***Action*** (siehe Unterkapitel 3.3.1.1). Die abgerundete Rechteckform bleibt erhalten, das Rechteck wird dabei aber in zwei horizontale Bereiche ohne sichtbaren Trenner aufgespalten (siehe Abb. 3.3). In der linken oberen Ecke steht der Bezeichner, daneben stehen optionale Zusatzinformationen welche hier nicht weiter aufgeführt werden da sie für die Geschäftsprozessmodellierung in *SOAMIG* nicht benötigt werden. Im unteren Teil der ***Activity*** befinden sich die Notationselemente welche dessen Verhalten festlegen.



Darstellung einer in ein Aktivitätsdiagramm einbettbaren ***Activity*** mit offenliegendem, inneren Aufbau

Activity als Aktivitätsdiagramm mit Notationselementen

Abbildung 3.3.: Graphische Darstellung von ***Activities***

Eine in ein Aktivitätsdiagramm eingesetzte ***Activity*** verhält sich nach außen wie eine ***Action***. Sie kann Daten, welche sie über ***Object Nodes*** (siehe Unterkapitel 3.3.3) und ***Object Flows*** (siehe Unterkapitel 3.3.2.1) zugeführt bekommt, bearbeiten sowie alte und neu produzierte Datensätze abführen. ***Activities*** werden, wie ***Actions***, über ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) in den Ausführungsfluss eingebunden. Die ***Activity***, bzw. das ihr zugrundeliegende Aktivitätsdiagramm, ist in sich abgeschlossen (vgl. [OMG09b], S. 318), d.h. es gibt keine ***Activity Edges*** die

Notationselemente in der ***Activity*** mit Notationselementen außerhalb der ***Activity*** verbinden. Ein- und ausgehende ***Activity Edges*** starten bzw. enden am Rand der ***Activity*** oder an dort angebrachten ***Object Nodes***.

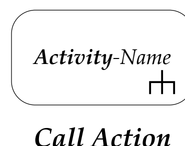
Das einer ***Activity*** zugrunde liegende Aktivitätsdiagramm wird graphisch als Rechteck dargestellt, welches alle Notationselemente umschließt (siehe Abb. 3.3). In der linken oberen Ecke befindet sich der Name in einem kleinen Fünfeck. Jedes Aktivitätsdiagramm wird als separates Modell notiert.

3.3.1.3. *Call Action*

Eine ***Call Behavior Action*** oder kurz ***Call Action*** wird als Platzhalter für eine ***Activity*** (siehe Unterkapitel 3.3.1.2) in ein Aktivitätsdiagramm eingesetzt, deren Inhalt u.a. nicht modelliert oder unwesentlich für das Verständnis des Aktivitätsdiagrammes ist. Die ***Call Action*** teilt sich den Namen mit der ***Activity***, deren Verhalten sie bei ihrer Ausführung aufruft.

Bei einem Verhaltensaufruf handelt es sich in der *UML* um eine atomare Tätigkeit, weshalb eine ***Call Action*** eine spezielle Sorte einer ***Action*** (siehe Unterkapitel 3.3.1.1) ist, woraus sich auch die Namensgleichheit ergibt (vgl. [OMG09b]). Aufgrund ihrer Abstammung teilt sich eine ***Call Action*** die Eigenschaften einer ***Action***, sie wird über ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) in den Ausführungsfluss eingebunden und kann über ***Object Nodes*** (siehe Unterkapitel 3.3.3) und ***Object Flows*** (siehe Unterkapitel 3.3.2.1) Daten erhalten, an die aufgerufene ***Activity*** zur Bearbeitung weitergeben und von der ***Activity*** produzierte Daten weiterreichen.

Graphisch unterscheidet sich eine ***Call Action*** von einer ***Action*** einzig durch das Gabelsymbol in der rechten unteren Ecke des abgerundeten Rechtecks (siehe Abb. 3.4). Der innere Aufbau der referenzierten ***Activity*** wird nicht in der ***Call Action*** abgebildet.



Call Action

Abbildung 3.4.: Graphische Darstellung einer ***Call Action***

3.3.2. *Flow Controls*

Diese Kategorie enthält das Werkzeug um den Start, die Steuerung und die Terminierung des Ablaufs eines Aktivitätsdiagrammes zu regulieren. Es wird unterschieden zwischen:

- ***Activity Edges*** (siehe Unterkapitel 3.3.2.1),
- ***Events*** (siehe Unterkapitel 3.3.2.2) und
- ***Control Nodes*** (siehe Unterkapitel 3.3.2.3).

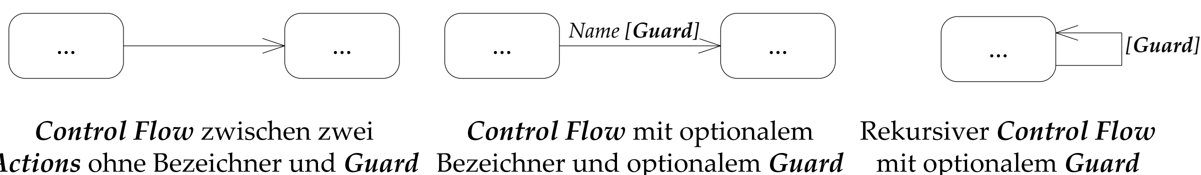
3.3.2.1. *Activity Edges*

Activity Edges modellieren gerichtete Verbindungen zwischen ***Activity Nodes*** (siehe Unterkapitel 3.3.1), ***Events*** (siehe Unterkapitel 3.3.2.2) sowie ***Control Nodes*** (siehe Unterkapitel 3.3.2.3) und legen die Ablaufreihenfolge eines Aktivitätsdiagrammes fest (vgl. [Kec06]). Die *UML* Spezifikation unterscheidet zwischen zwei ***Activity Edge*** Typen, den ***Control Flows*** und den ***Object Flows***. ***Object Flows*** unterscheiden sich nur dahingegen von ***Control Flows***, dass sie ***Activity Nodes***, ***Events*** und ***Control Nodes*** nicht direkt, sondern indirekt über an diese angebrachte ***Object Nodes*** (siehe Unterkapitel 3.3.3) verbinden. Zusätzlich zum Ausführungsfluss modellieren ***Object Flows*** auch den Datenfluss im Aktivitätsdiagramm, indem sie Datensätze (***Objects***) zwischen ***Object Nodes*** transportieren. Da ansonsten keine Unterschiede bestehen, beziehen sich die weiteren Beschreibungen auf beide ***Activity Edge*** Typen.

Graphisch werden ***Activity Edges*** als durchgehende Linie mit einer offenen Pfeilspitze an der Seite des Zieles dargestellt (siehe Abb. 3.5 für ***Control Flows*** und Abb. 3.6 für ***Object Flows***). Eine ***Activity Edge*** kann optional mit einem Bezeichner nahe der Pfeilspitze versehen werden (siehe Abb. 3.5 für ***Control Flows*** und Abb. 3.6 für ***Object Flows***).

Zusätzlich ist die Angabe eines ***Guards*** möglich, eines formalen oder informalen Ausdrucks welcher während der Laufzeit ausgewertet wird und überprüft ob eine bestimmte Bedingung im Aktivitätsdiagramm erfüllt wird. Ergibt die Auswertung des Ausdrucks den boolschen Wahrheitswert *true*, läuft der Ausführungsfluss über diese ***Activity Edge*** weiter. Andernfalls wird die ***Activity Edge*** nicht durchlaufen. Graphisch wird der Ausdruck eines ***Guards*** in eckigen Klammern hinter den Bezeichner einer ***Activity Edge*** notiert (siehe Abb. 3.5 für ***Control Flows*** und Abb. 3.6 für ***Object Flows***).

Ein ***Activity Edge*** hat immer genau eine Quelle und ein Ziel. Ein ***Activity Edge*** kann dabei auch rekursiv sein. Hierbei ist, sofern nicht beabsichtigt, ein ***Guard*** verpflichtend, da sonst eine Endlosschleife entsteht (siehe Abb. 3.5 für ***Control Flows*** und Abb. 3.6 für ***Object Flows***).



Control Flow zwischen zwei ***Actions*** ohne Bezeichner und ***Guard*** — ***Control Flow*** mit optionalem Bezeichner und optionalem ***Guard*** — Rekursiver ***Control Flow*** mit optionalem ***Guard***

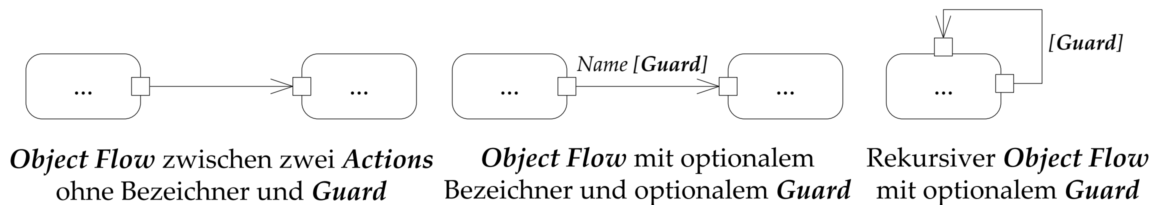
Abbildung 3.5.: Graphische Darstellung von ***Activity Edges*** des Typs ***Control Flow***

Activity Nodes, ***Events*** und ***Control Nodes*** können, sofern nicht anders angegeben, mehrere ein- und ausgehende ***Activity Edges*** besitzen. Hierbei stellen mehrere eingehende ***Activity Edges*** in einen ***Activity Node***, eine ***Send Signal Action*** (siehe Unterkapitel 3.3.2.2) oder eine ***Accept Signal Action*** (siehe Unterkapitel 3.3.2.2) eine implizite ***AND***-Verknüpfung dar, d.h. mit der Ausführung des Ziels wird gewartet, bis der Ausführungsfluss das Ziel über alle eingehenden ***Activity Edges*** erreicht hat. Dieses Konzept ist vergleichbar mit einem ***Join Node*** (siehe Unterkapitel 3.3.2.3).

Bei ***Final Nodes*** (siehe Unterkapitel 3.3.2.3) stellen hingegen mehrere eingehende ***Activity Edges*** eine implizite ***OR***-Verknüpfung dar, d.h. die Ausführung eines ***Final Nodes*** beginnt, sowohl bei Eingang des Ausführungsflusses über eine ***Activity Edge***, als auch beim Eingang mehrerer Ausführungsflüsse über mehrere oder alle ***Activity Edges*** zugleich.

Ein als gesamtes Modell dargestelltes *UML* Aktivitätsdiagramm besitzt keine ein- und keine ausgehenden ***Activity Edges***.

Schließlich stellen mehrere, von ***Activities***, ***Initial Nodes***, ***Send Signal Actions***, ***Accept Signal Actions*** ausgehende ***Activity Edges*** ein implizites, den Ausführungsfluss aufspaltendes ***AND*** dar, d.h. der Ausführungfluss wird auf mehrere unabhängige, parallel verlaufende Flüsse aufgeteilt. Dieses Konzept ist vergleichbar mit einem **Fork Node** (siehe Unterkapitel 3.3.2.3).



Object Flow zwischen zwei ***Actions*** ohne Bezeichner und ***Guard*** | ***Object Flow*** mit optionalem Bezeichner und optionalem ***Guard*** | Rekursiver ***Object Flow*** mit optionalem ***Guard***

Abbildung 3.6.: Graphische Darstellung von ***Activity Edges*** des Typs ***Object Flow***

3.3.2.2. *Events*

Events stellen das Auftreten eines von außerhalb oder innerhalb des Aktivitätsdiagrammes eingeläuteten Herganges dar, der sich auf den Ausführungfluss auswirkt, z.Bsp. das Eintreffen eines Signales wie die Kundenanfrage im Einführungsbeispiel (siehe Abb. 3.1). Es wird unterschieden zwischen den Unterkategorien:

- ***Initial Node***, ***Activity Final Node*** und ***Flow Final Node*** sowie
- ***Send Signal Action*** und ***Accept Signal Action***.

Auch wenn die im folgenden vorgestellten Elemente in der *UML*-Spezifikation nicht unter dem Begriff ***Event*** zusammengefasst vorzufinden sind, teilen sie sich ähnliche Zielsetzungen, weshalb sie hier zusammen auftreten. Diese Klassifizierung ist an die Kategorisierung von ***Events*** aus der *BPMN* angelehnt (siehe Unterkapitel 4.3.2.2).

Initial Node, *Activity Final Node* und *Flow Final Node*

Ein ***Initial Node*** legt die Stelle im Aktivitätsdiagram fest, an der der Ausführungsfluss beginnt. Besitzt ein Diagramm mehrere ***Initial Nodes*** werden bei Ausführungsbeginn voneinander unabhängig laufende Flüsse gestartet. Besitzt ein Aktivitätsdiagramm hingegen kein ***Initial Node***,

werden die Stellen an denen die Ausführung beginnt durch ***Accept Signal Actions*** oder ***Object Nodes*** (siehe Unterkapitel 3.3.3) festgelegt. Ein **Initial Node** besitzt keine ein- und genau einen ausgehenden ***Control Flow*** (siehe Unterkapitel 3.3.2.1), ein ausgehender ***Object Flow*** (siehe Unterkapitel 3.3.2.1) ist nicht erlaubt (vgl. [OMG09b]).

Activity Final Node und ***Flow Final Node*** bilden die Gegenstücke zum ***Initial Node***. Beide modellieren die geplante Beendigung eines Ausführungsflusses und können zusammen und in mehreren Ausführungen in einem Aktivitätsdiagramm vorkommen (vgl. [Kec06]). Während bei Eintritt des Ausführungsflusses in ein ***Activity Final Node*** alle Ausführungsflüsse im Aktivitätsdiagramm enden und damit die gesamte Ausführung des Diagrammes terminiert, beendet ein ***Flow Final Node*** nur den hineinlaufenden Fluss. ***Activity Final Nodes*** und ***Flow Final Nodes*** besitzen genau eine ein- und keine ausgehenden ***Control Flows***, ein eingehender ***Object Flow*** ist nicht erlaubt (vgl.**[OMG09b]**).

Graphisch wird ein ***Initial Node*** als ausgefüllter Kreis dargestellt. Ein ***Activity Final Node*** wird dagegen als ausgefüllter Kreis innerhalb eines größeren Kreises mit einfacher Umrandung notiert und ein ***Flow Final Node*** besteht aus einem Kreis mit einfacher Umrandung mit einem Kreuz im Inneren. Optional lässt sich für jedes der drei Notationselemente ein frei wählbarer Bezeichner definieren, welcher unmittelbar an der Kreisaußenseite (des äußeren Kreises) angebracht wird (siehe Abb. 3.7).

Initial Node mit optionalem Bezeichner	***Activity Final Node*** mit optionalem Bezeichner	***Flow Final Node*** mit optionalem Bezeichner

Abbildung 3.7.: Graphische Darstellung eines ***Initial Nodes***, eines ***Activity Final Nodes*** und eines ***Flow Final Nodes***

Send Signal Action und ***Accept Signal Action***

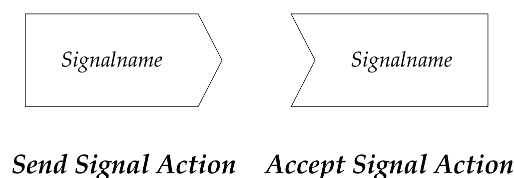
Erreicht der Ausführungsfluss eine ***Send Signal Action*** wird ein asynchrones Signal an ein oder mehrere ***Accept Signal Actions*** geschickt, welche nur auf dieses Signal reagieren. Nach Auslösen des Signals läuft der Ausführungsfluss am ***Send Signal Action*** weiter, parallel werden weitere, unabhängige Ausführungsflüsse an den ***Accept Signal Actions*** gestartet. Eine ***Accept Signal Action*** kann einen Signal auch mehrmals auswerten.

Die Ziele/ Quellen einer ***Send Signal Action*** bzw. einer ***Accept Signal Action*** müssen nicht zwingend in der gleichen ***Activity*** (siehe Unterkapitel 3.3.1.2) oder dem gleichen Aktivitätsdiagramm liegen, sie brauchen auch nicht im aktuell modellierten Aktivitätsdiagramm abgebildet sein. Wird ein Aktivitätsdiagramm als ***Activity*** in ein weiteres Aktivitätsdiagramm eingebettet, sind automatisch alle ***Send Signal Actions*** und ***Accept Signal Actions*** mit gleichem (Signal-)Namen miteinander verbunden.

Bei Signalsendung und -empfang handelt es sich nach der *UML*-Spezifikation um atomare Tätigkeiten, weshalb beide Notationselemente spezielle ***Actions*** sind – neben der Namensgleichheit erben sie auch die Eigenschaften, erweitern diese aber leicht. Hat eine ***Accept Signal Action*** eingehende ***Activity Edges*** (siehe Unterkapitel 3.3.2.1), wird sie für den Empfang von Signalen aktiviert, sobald sie vom Ablauf erreicht wird. Andernfalls ist sie bereits bei Ausführungsbeginn des Aktivitätsdiagrammes bereit Signale zu empfangen (vgl. [Kec06]) und kann im Gegensatz zu einer ***Action*** auch als Startpunkt eines Aktivitätsdiagrammes dienen.

Die graphische Notation einer ***Send Signal Action*** entspricht einem konvexen Fünfeck , die einer ***Accept Signal Action*** hingegen einem konkaven Fünfeck (siehe Abb. 3.2).

Mit einer ***Send Signal Action*** und einer ***Accept Signal Action*** kann beispielsweise einem Geschäftsprozessakteur mitgeteilt werden, die Abarbeitung bestimmer Tätigkeiten, die im Zusammenhang mit den Ereignissen die zum auslösen des Triggers durch einen weiteren Akteur führten, aufzunehmen. In Bezug auf das Notationsbeispiel am Anfang dieses Kapitels (siehe Unterkapitel 3.2) beschreibt beispielsweise die Anfrage eines Kunden an den Reisebüroangestellten eine ***Send Signal Action*** und die Bearbeitung des Signals durch den Angestellten das ***Accept Signal Action*** *receiveCustomerRequest*.



Send Signal Action ***Accept Signal Action***

Abbildung 3.8.: Graphische Darstellung einer ***Send Signal Action*** und einer ***Accept Signal Action***

3.3.2.3. *Control Nodes*

Control Nodes sind für die Verzweigung (***Branch***) und Zusammenführung (***Merge***) von ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) zuständig und koordinieren den Ausführungsfluss eines Aktivitätsdiagrammes. Unterschieden wird zwischen den folgenden Unterkategorien:

- ***Decision Node*** und ***Merge Node*** sowie
- ***Fork Node*** und ***Join Node***.

Decision Node und *Merge Node*

Decision Nodes und ***Merge Nodes*** dienen der Modellierung alternativer Pfade – *XOR*-Verknüpfung. An einem ***Decision Node*** wird der eingehende Ausführungsfluss anhand der Auswertung disjunkter ***Guards*** an den ausgehenden ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) auf genau einen

Pfad gelenkt und an einem ***Merge Node*** fließt der Ausführungsfluss über genau einen der eingehenden Pfade/ ***Activity Edges*** ein. Die ein- und ausgehenden ***Activity Edges*** eines ***Decision Nodes/ Merge Nodes*** müssen vom gleichen Typ sein, also alle ***Control Flows*** oder ***Object Flows*** sein (siehe Unterkapitel 3.3.2.1 für ***Control Flows*** und ***Object Flows***)(vgl. [OMG09b]).

Obwohl die *UML* Spezifikation die Zusammenlegung aufeinanderfolgender ***Decision Nodes*** und ***Merge Nodes*** als ein Symbol mit mehreren ein- und mehreren ausgehenden ***Activity Edges*** erlaubt, ist die **Zusammenlegung aus Gründen der besseren Unterscheidung der Notationselemente in dieser Arbeit nicht vorgesehen**. Ein ***Decision Node*** besitzt also immer genau einen ein- und mindestens zwei ausgehende ***Activity Edges***, ein ***Join Node*** hingegen immer mindestens zwei ein- und genau einen ausgehenden ***Activity Edge***.

Graphisch werden ***Decision Nodes*** und ***Merge Nodes*** durch eine leere Raute dargestellt (siehe Abb. 3.9). Sie unterscheiden sich nur durch die Anzahl ihrer ein- und ausgehenden ***Activity Edges***. ***Decision Nodes*** und ***Merge Nodes*** wird kein Bezeichner zugewiesen (vgl. [OMG09b]).

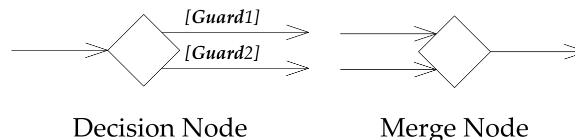


Abbildung 3.9.: Graphische Darstellung eines ***Decision Nodes*** und eines ***Merge Nodes***

Fork Node und Join Node

Fork Nodes teilen den Ausführungsfluss in mehrere nebenläufige Flüsse, die unabhängig voneinander abgearbeitet werden (vgl. [Kec06]). Im Gegensatz dazu fassen ***Join Nodes*** mehrere Ausführungsflüsse zu einem einzigen Fluss zusammen, indem sie den sie verlassenden Ausführungsfluss erst freigeben, nachdem alle in sie eingehende Flüsse an ihr angelangt sind – *AND*-Verknüpfung (vgl. ebd.). Die ein- und ausgehenden ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) eines ***Fork Nodes/ Join Nodes*** müssen vom gleichen Typ sein, also alle ***Control Flows*** oder ***Object Flows*** sein (siehe Unterkapitel 3.3.2.1 für ***Control Flows*** und ***Object Flows***)(vgl. [OMG09b]).

Für ***Join Nodes*** lassen sich andere Vereinigungstypen (***Join Specifications***) als die logische *AND*-Verknüpfung definieren. Die formale oder informale Angabe erfolgt in geschweiften Klammern und wird von einem Gleichheitszeichen und dem Schlüsselwort ***joinSpec*** angeführt. Beispielsweise lässt sich eine inklusive Vereinigung mit dem ***JoinSpec***-Ausdruck *{joinSpec = A or B}* realisieren, wobei *A* und *B* die Namen der eingehenden ***Activity Edges*** eines ***Join Nodes*** darstellen.

Obwohl die *UML* Spezifikation die Zusammenlegung aufeinanderfolgender ***Fork Nodes*** und ***Join Nodes*** als ein Symbol mit mehreren ein- und mehreren ausgehenden ***Activity Edges*** erlaubt, ist die **Zusammenlegung aus Gründen der besseren Unterscheidung der Notationselemente in dieser Arbeit nicht vorgesehen**. Ein ***Fork Node*** besitzt also immer genau einen ein- und mindestens zwei

ausgehende ***Activity Edges***, ein ***Join Node*** hingegen immer mindestens zwei ein- und genau einen ausgehenden ***Activity Edge***.

Die graphische Darstellung eines ***Join Nodes*** ist mit der eines ***Fork Nodes*** identisch, sie werden als Balken dargestellt (siehe Abb. 3.9). Bei ***Fork Nodes*** wird die optionale ***Join Specification*** je nach Ausrichtung des graphischen Elementes darüber oder daneben geschrieben (siehe Abb. 3.9).

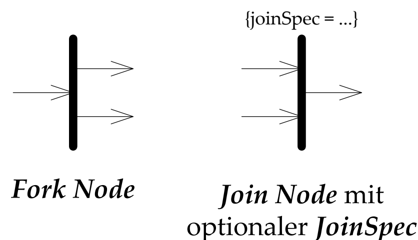


Abbildung 3.10.: Graphische Darstellung eines ***Fork Nodes*** und eines ***Join Nodes***

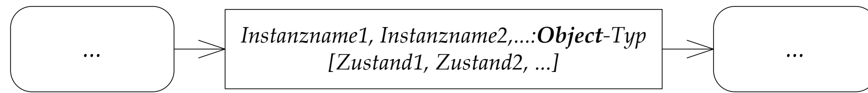
3.3.3. *Resources – Objects*

Object Nodes modellieren die Übergabe von Daten (***Objects***/ ***Resources***) zwischen ***Activity Nodes*** (siehe Unterkapitel 3.3.1). Bei ***Objects*** handelt es sich um Daten im Sinne des objektorientierten Ansatzes (vgl. [Sta06]). Ein ***Object*** stellt eine Instanz eines Typs dar, welcher durch eine *UML*-Klasse abgebildet wird und Spezialisierungen oder Generalisierungen haben kann. ***Objects*** werden bei der Ausführung von ***Activity Nodes*** erzeugt und verarbeitet, können einem Geschäftsprozess – einer ***Activity*** (siehe Unterkapitel 3.3.1.2) oder einem *UML* Aktivitätsdiagramm – über ***Object Nodes*** aber auch von außerhalb zu- oder nach außen abgeführt werden. Die ***Object***-Weitergabe erfolgt über ***Object Flows*** (siehe Unterkapitel 3.3.2.1). Ein ***Object Node*** kann auch mehr als ein ***Object*** eines Typs enthalten und ein ***Object Flow*** kann mehr als ein ***Object*** eines Typs oder verschiedener Typen transportieren.

Ein ***Object*** besitzt einen Bezeichner und einen optionalen Zustand, der sich bei der Bearbeitung eines ***Objects*** durch einen ***Activity Node*** ändern kann. Der Bezeichner setzt sich aus einem Instanznamen und einem ***Object***-Typ zusammen. Alternativ kann der ***Object***-Typ oder der Instanzname weggelassen werden.

Graphisch können ***Objects Nodes*** und ***Objects*** auf zwei Arten dargestellt werden, in Rechteck- oder in *Pin*-Notation:

- bei der Rechtecknotation wird ein ***Object Node*** als leeres Reckteck mit einfacher Umrahmung zwischen ein- und ausgehenden, ***Objects***-transportierende ***Object Flows*** dargestellt. Die ***Objects*** selbst werden nicht graphisch dargestellt, es werden lediglich die Bezeichner und Zustände der einzelnen Objects in das Rechteck geschrieben (siehe Abb. 3.11).
- bei der *Pin*-Noation wird ein ***Object Node*** als kleines Quadrat am Rand eines ***Activity Nodes*** notiert. Die ***Objects*** selbst werden nicht graphisch dargestellt, es werden lediglich die Bezeichner und Zustände der einzelnen ***Objects*** an die Pins geschrieben (siehe Abb. 3.12).

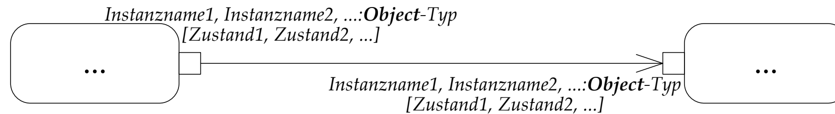


Object Node in Recktecknotation mit optionalen Instanznamen
und optionalen Zustandsangaben zwischen zwei ***Actions***

Abbildung 3.11.: Graphische Darstellung eines ***Object Nodes*** **in Rechtecknotation**

Die Unterschiede zwischen Reckteck- und *Pin*-Notation sind nicht nur graphischer Natur:

- bei der Übergabe eines ***Objects*** anhand der Rechtecknotation wird vorgegeben, dass der Ziel-***Activity Node*** nur genau diesen ***Object***-Typen über den mit dem ***Object Node*** verbundenen, eingehenden ***Object Flow*** empfangen kann (siehe Abb. 3.11).
- bei der *Pin*-Notation kann auf aus- und eingehender Seite ein unterschiedlicher ***Object***-Typ angegeben sein. Hierbei muss auf die Kompatibilität des Typs geachtet werden, ein eingehendes ***Object*** muss von einem Untertyp des angegebenen ***Object***-Typs oder von diesem Typ selbst stammen (siehe Abb. 3.12).



Object Node in *Pin*-Notation mit optionalen Instanznamen und
optionalen Zustandsangaben an zwei ***Actions***

Abbildung 3.12.: Graphische Darstellung eines ***Object Nodes*** **in *Pin*-Notation**

3.3.4. *Participant Partitions – Activity Partitions*

Ein Akteur (***Participant***) stellt eine Organisationseinheit in einem Geschäftsprozess dar, dabei handelt es sich um einen Typ oder eine Rolle die u.a. eine Firma, ein System, eine Personengruppe oder ein einzelnes Individuum bei der Ausführung des Prozesses einnehmen kann (vgl. [Kec06]). In *UML* Aktivitätsdiagrammen wird der Verantwortungsbereich einer Organisationseinheit als ***Activity Partition*** modelliert. Alle enthaltenen ***Activity Nodes*** (siehe Unterkapitel 3.3.1) werden bei der Ausführung des Diagramms durch diesen Akteur ausgeführt. Die Modellierung einer ***Activity Partition*** in einem Aktivitätsdiagramm ist optional.

Graphisch wird eine ***Activitiy Partition*** als offenes Rechteck dargestellt. Der Bezeichner wird im Inneren des Rechtecks in einem weiteren Rechteck notiert das je nach Ausrichtung der ***Activity Partition*** an der oberen oder linken Rechteckbegrenzungslinie des äußeren Rechtecks angrenzt (siehe Abb. 3.13). ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) verbinden ***Activity Nodes***, ***Ev*****ents** (siehe Unterkapitel 3.3.2.2) und ***Control Nodes*** (siehe Unterkapitel 3.3.2.3) eines Aktivitätsdiagramms über die Grenzen von ***Activitiy Partitions*** hinweg.

Activity Partitions können geschachtelt werden, womit sich die Zuständigkeitsbereiche auf mehrere Untereinheiten aufteilen lassen (siehe Abb. 3.13). Diese Untereinheiten sind nur innerhalb der Obereinheit für die modellierten Tätigkeiten zuständig (vgl. [Kec06]). Außerdem können sich die Verantwortungsbereiche auch kreuzen, was bedeutet dass sie sich die Durchführung der enthaltenen Tätigkeiten teilen (siehe Abb. 3.13).

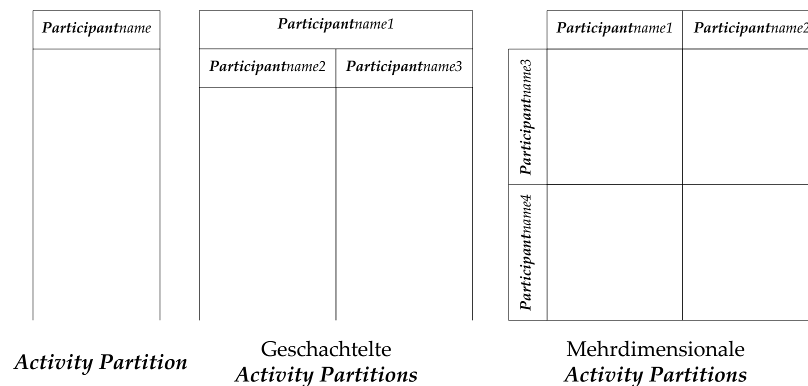


Abbildung 3.13.: Graphische Darstellung von ***Activity Partitions***

3.3.5. *Annotations – Comments*

Ein ***Comment*** wird eingesetzt, um weitere Informationen über Notationselemente, den Ausführungsfluss, oder Implementierungsdetails anzugeben. Es lässt sich aber prinzipiell jede beliebige Anmerkung (***Annotation***) ausdrücken. ***Comments*** können über eine ungerichtete Verbindung an beliebige Notationselement geheftet werden oder frei stehen, dann gehören sie implizit zu dem Aktivitätsdiagramm/ der ***Activity*** (siehe Unterkapitel 3.3.1.2) in dem/ der sie sich befinden.

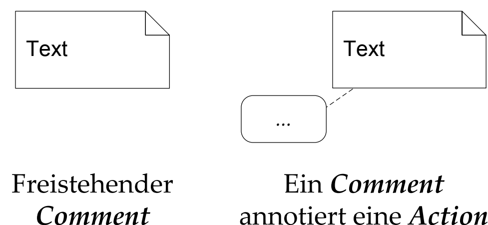


Abbildung 3.14.: Graphische Darstellung von ***Comments***

Die Darstellung eines ***Comments*** besteht aus einem Rechteck mit nach innen geknickter, oberen rechten Ecke und beliebigem, formalen oder informalen, Inhalt (siehe Abb. 3.14). Ist ein ***Comment*** an ein Element gebunden, wird diese Verbindung graphisch durch eine ungerichtete, gestrichelte Linie dargestellt (siehe Abb. 3.14).

4. *Business Process Modeling Notation* (*BPMN*)

4.1. Einführung in die *BPMN*

Der Akzent auf dem Gebiet der Geschäftsprozessmodellierungssprachen lag in den späten 90er Jahren des letzten Jahrhunderts und zu Beginn des 21. Jahrhunderts vorwiegend auf der Entwicklung formaler Notationen, welche auf die Ausführbarkeit in *Business Process Management Systemen* (*BPMS*) hin optimiert waren. Mit diesen Sprachen erstellte Geschäftsprozessmodelle waren jedoch für Leser schwer verständlich, weshalb die *Business Process Management Initiative*[1] (*BPMI*), welche 2005 in die *Object Management Group* (*O*MG) aufging, die Entwicklung der *Business Process Management Notation* (BPMN) in Auftrag gab. Unter der Federführung von Steven A. WHITE der Firma *IBM* sollte die Lücke zwischen einer formalen, maschinennahen Notation und einer für Menschen leicht verständlichen Dastellung überbrückt werden. Hierfür orientierten sich die Entwickler stark an *Flowcharts*, deren Einsatz bis heute weit verbreitet ist. Aus dieser Entwicklung ging 2006 die Version 1.0 der *BPMN* als *OMG*-Standard hervor.

Das Hauptziel bei der Entwicklung war, eine nicht überladene und leicht nachvollziehbare, graphische Notation für Geschäftsprozessmodelle. Dementsprechend beinhaltet die Sprachspezifikation eine detaillierte Beschreibung der Sprachsyntax und der graphischen Notation. Die Sprachsemantik wird dabei aber nur grob und informal festgelegt. Ein offizielles Metamodell der Sprache existiert nicht. Dies bereitet bis heute Entwicklern von Ausführungssprachen, welche eine Abbildung der *BPMN* auf ihre Notation vorbereiten, Schwierigkeiten. Die unpräzise Spezifikation und unvollständige Semantik fordert ein hohes Maß an Eigeninterpretation, was zu Inkompatibiliäten zwischen den Ansätzen verschiedener Hersteller führt. Auch die bis heute aktuelle Version 1.2 und die Vorgängerversion 1.1 ändern nichts an diesem Missstand. Erst für die 2010 erwartete Version 2.0 stellt die *OMG* u.a. eine formale, semantische Spezifikation und ein Metamodell in Aussicht (vgl. [All08]).

Abgesehen von diesen für einen Standard schwerwiegenden Benachteiligungen steht die *BPMN* den *UML* Aktivitätsdiagrammen aber in nichts nach und bietet eine solide und leicht zugängliche Notation für die Modellierung von Geschäftsprozessen. Für die Aufnahme in diese Arbeit und das *SOAMIG*-Projekt spricht zudem die breite Akzeptanz und Verbreitung der *BPMN* über mehrere Geschäftsfelder hinweg (vgl. [All08]).

[1] `http://www.bpmi.org/`

4.2. Notationsübersicht

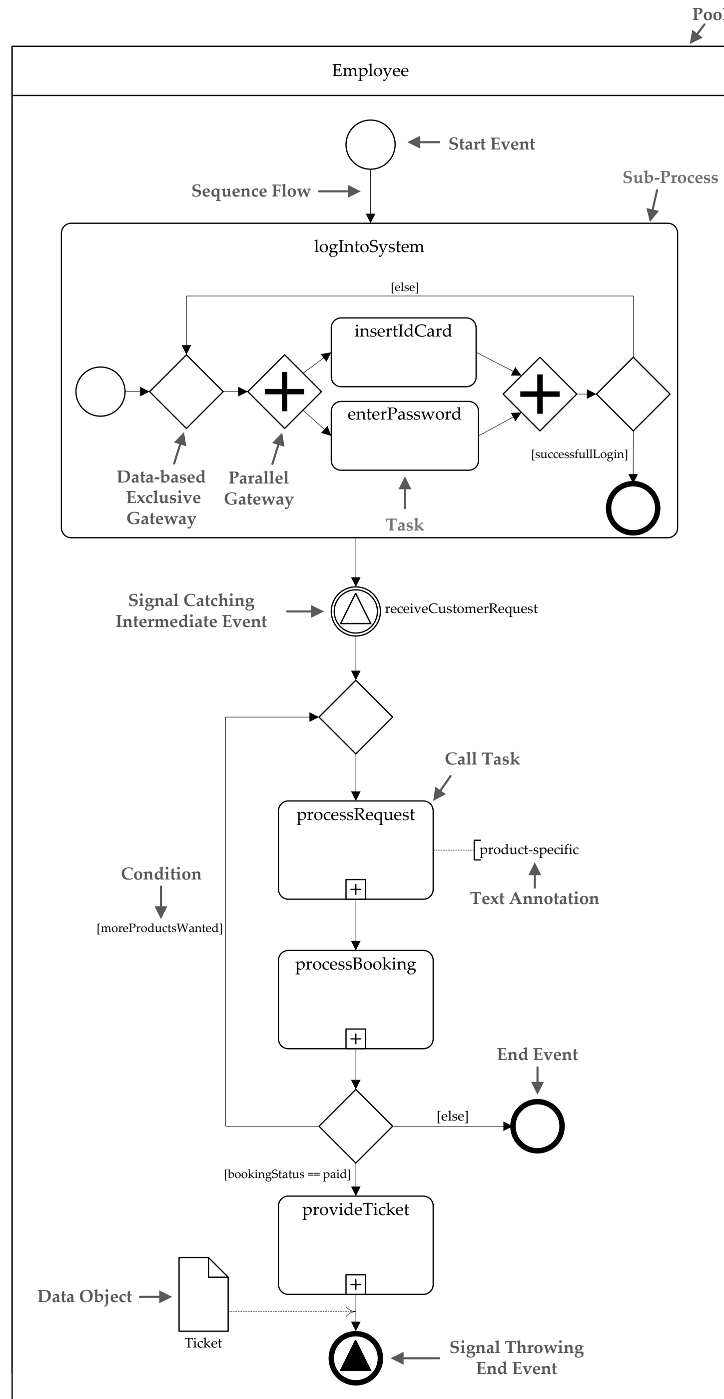


Abbildung 4.1.: Modellierung des Geschäftsprozesses *sellTrainticket* als *BPMN* Diagramm

Zur Einführung in die *BPMN* wird in Abbildung 4.1 die Modellierung des Geschäftsprozesses *sell-Trainticket* dargestellt, welcher bereits bei der Notationsübersicht der *UML* Aktivitätsdiagramme beschrieben wurde (siehe Unterkapitel 3.2). Unterschiede in der graphischen Repräsentation ergeben sich durch den Einsatz unterschiedlicher Notationen, doch die beiden Diagramme lassen sich sehr ähnlich lesen und mit dem Verständnis der jeweils anderen Darstellung nachvollziehen.

Die Semantik des Geschäftsprozesses wurde bei der Abbildung in die *BPMN* erhalten: ein Angestellter eines Reisebürounternehmens meldet sich mit Password und Identifikationskarte an seinem Computersystem an und ist dann bereit Kundenanträge zu bearbeiten, zu buchen und dem Kunden die bezahlten Fahrkarten auszuhändigen.

Ein auffälliger Unterschied zwischen beiden Notationen ergibt sich bereits bei der Darstellung des Diagrammes. Während ein *UML* Aktivitätsdiagramm eine spezielle graphische Darstellung besitzt, wird bei der *BPMN* auf den graphischen Vermerk des abgebildeten Geschäftsprozesses verzichtet. Dagegen sind die Repräsentationen der, an der Ausführung des Geschäftsprozesses teilnehmenden Akteurs (*Employee*), nahezu identisch: ***Activity Partitions*** bei *UML* Aktivitätsdiagrammen (siehe Unterkapitel 3.3.4) und ***Pools*** in der *BPMN* (siehe Unterkapitel 4.3.4) ähneln sich nicht nur graphisch, auch semantisch verfolgen sie ähnliche Konzepte.

Ähnlichkeiten gibt es auch bei *BPMN* ***Events***: ***Start Events***, **End Events**, ***Signal Catching Events*** und ***Signal Throwing Events*** haben Gemeinsamkeiten mit ***Initial Nodes***, ***Activity Final Nodes***, ***Send Signal Actions*** und ***Accept Signal Actions*** (vgl. Unterkapitel 3.3.2.2 mit Unterkapitel 4.3.2.2). Es handelt sich um das Starten und Terminieren des Geschäftsprozesses sowie das Senden und Empfangen von Signalen zum auslösen von Arbeitsabläufen.

Auch bei den Konzepten nicht weiter unterteilbarer Tätigkeiten (***Tasks***, siehe Unterkapitel 4.3.1.1), zusammengesetzter Tätigkeiten (***Sub-Processes***, siehe Unterkapitel 4.3.1.2) und den Aufrufen von ***Sub-Processes*** durch ***Call Tasks*** (siehe Unterkapitel 4.3.1.3), sind die Parallelen zu *UML* ***Actions*** (siehe Unterkapitel 3.3.1.1), ***Activities*** (siehe Unterkapitel 3.3.1.2) und ***Call Actions*** (siehe Unterkapitel 3.3.1.3) nicht von der Hand zu weisen. Die Aufteilung (***Branch***) und Zusammenführung (***Merge***) von Ausführungsflüssen übernehmen ***Gateways*** (siehe Unterkapitel 4.3.2.3) wie u.a. die im Beispiel vorkommenden ***Data-based Exclusive Gateways*** und ***Parallel Gateways***, welche in *UML* unter dem Punkt ***Control Nodes*** (siehe Unterkapitel 3.3.2.3) wiederzufinden sind. Anmerkungen und Kommentare u.a. zum Geschäftsprozess oder zum Ablauf lassen sich analog zu *UML* ***Comments*** als *BPMN* ***Text Annotations*** modellieren (vgl. 3.3.5 mit Unterkapitel 4.3.5).

Kleinere Unterschiede gibt es bei den Verbindungen der Notationselemente durch ***Sequence Flows*** (siehe Unterkapitel 4.3.2.1) bei der *BPMN* und ***Control Flows*** (siehe Unterkapitel 3.3.2.1) bei den *UML* Aktivitätsdiagrammen. Während beide für die Steuerung der Ausführung zuständig sind, wird in der *BPMN*, nicht wie bei der *UML*, zwischen ***Control Flow*** und ***Object Flow*** unterschieden (siehe Unterkapitel 3.3.2.1), d.h. die in der *BPMN* ***Data Object*** (siehe Unterkapitel 4.3.3) genannten Datensätze werden direkt von ***Sequence Flows*** transportiert (siehe Unterkapitel 4.3.2.1). Zudem eignen sich ***Sequence Flows*** nur als Kanten zwischen Tätigkeiten im Verantwortungsbereich des gleichen Akteurs, für die Kommunikation zwischen verschiedenen Pools sind ***Message Flows*** vorgesehen (siehe Unterkapitel 4.3.2.1).

4.3. Notationselemente

Im folgenden werden die Sprachkonzepte und Notationselemente der *BPMN* beschrieben, die sich in die in der Einleitung zu diesem Kapitel (siehe Kapitel 2) identifizierten Elementgruppen einordnen lassen und für die Erhebung von Geschäftsprozessmodellen im *SOAMIG*-Projekt benötigt werden. **Um eine möglichst große Überschneidung bei den Modellierungsmöglichkeiten zu erhalten, wurden die im folgenden vorgestellten Elemente in Anlehnung an Sprachelemente aus den *UML* Aktivitätsdiagrammen ausgewählt.** Im Laufe der Beschreibungen wird auf das jeweilige Gegenstück aus den *UML* Aktivitätsdiagrammen hingewiesen. Die Elemente sind den folgenden fünf Kategorien zugeordnet:

- ***Activities*** (siehe Unterkapitel 4.3.1),
- ***Flow Controls*** (siehe Unterkapitel 4.3.2),
- ***Resources – Data Objects*** (siehe Unterkapitel 4.3.3),
- ***Participant Partitions – Swimlanes*** (siehe Unterkapitel 4.3.4) und
- ***Annotations – Text Annotations*** (siehe Unterkapitel 4.3.5).

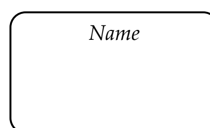
4.3.1. *Activities*

Diese Kategorie beinhaltet Sprachelemente zur Beschreibung des Verhaltens eines *BPMN* Diagrammes und bildet einfache und komplexe Arbeitsabläufe eines Geschäftsprozesses ab. Es wird unterschieden zwischen:

- ***Task*** (siehe Unterkapitel 4.3.1.1),
- ***Sub-Process*** (siehe Unterkapitel 4.3.1.2),
- ***Call Task*** (siehe Unterkapitel 4.3.1.3).

4.3.1.1. *Task*

Ein ***Task*** beschreibt einen elementaren, nicht weiter zerlegbaren oder nicht näher dargestellten Arbeitsablauf eines Prozesses und entspricht einer ***Action*** in *UML* Aktivitätsdiagrammen (siehe Unterkapitel 3.3.1.1).



Task

Abbildung 4.2.: Graphische Darstellung eines ***Tasks***

Ein ***Task*** wird über mindestens einen ein- und mindestens eine ausgehenden ***Sequence Flow*** (siehe Unterkapitel 4.3.2.1) in den Ausführungsfluss eingebunden, d.h. über gerichtete Verbindungen mit anderen Notationselementen verbunden. Die Ausführungsreihenfolge wird durch ***Flow Control*** Elemente bestimmt (siehe Unterkapitel 4.3.2). Über ***Data Objects*** (siehe Unterkapitel 4.3.3) und ***Sequence Flows*** können einem ***Task*** Daten zur Bearbeitung zugeführt und durch den ***Task*** produzierte Datensätze an weitere ***Activities*** weitergegeben werden.

Graphisch wird ein ***Task*** als Rechteck mit abgerundeten Ecken dargestellt (siehe Abb. 4.2). Der Name, eine beliebig detaillierte formale oder informale Beschreibung der Aufgabe die der ***Task*** erfüllt, wird oben mittig plaziert.

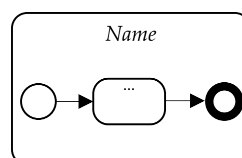
4.3.1.2. *Sub-Process*

Ein ***Sub-Process*** ist mit einer ***Activity*** aus *UML* Aktivitätsdiagrammen (siehe Unterkapitel 3.3.1.2) vergleichbar, stellt dementsprechen auch eine nicht-atomare Tätigkeit dar und beschreibt ein komplexes, detailliertes Verhalten in einem ihm zugrundeliegenden *BPMN* Diagramm. Jeder ***Sub-Process*** setzt sich aus anderen *BPMN* Notationselementen zusammen und kann weitere ***Sub-Processes*** enthalten. Ein ***Sub-Process*** kann nicht in sich selbst vorkommen.

Ein ***Sub-Process*** kann auf zwei Arten in ein *BPMN* Diagramm eingebunden werden:

- Wenn der innere Aufbau noch nicht feststeht oder für das Verständnis des *BPMN* Diagrammes unwesentlich ist, kann das Verhalten eines ***Sub-Processes*** über einen ***Call Task*** aufgerufen werden (siehe Unterkapitel 4.3.1.3). Der Inhalt des ***Sub-Processes*** wird nicht dargestellt.
- Steht der Inhalt des ***Sub-Processes*** fest und wird dessen komplette Darstellung in einem *BPMN* Diagramm benötigt, liegt der innere Aufbau offen, d.h. alle graphischen Elemente werden dargestellt. Graphisch unterscheidet sich ein solcher ***Sub-Process*** kaum von einem ***Task*** (siehe Unterkapitel 4.3.1.1). Die Rechteckform mit abgerundeten Ecken bleibt erhalten, zusätzlich zum Namen werden darunter noch die das Verhalten beschreibenden Elemente notiert.

In *BPMN* Diagramme eingesetzte ***Sub-Processes*** verhalten sich nach außen hin wie ***Tasks***. Sie werden jeweils über mindestens einen ein- und einen ausgehenden ***Sequence Flow*** (siehe Unterkapitel 4.3.2.1) in den Ausführungsfluss eingebunden worüber sie ***Data Objects*** (siehe Unterkapitel 4.3.3) erhalten und weitergeben können. Ein ***Sub-Process*** ist in sich abgeschlossen, d.h. es gibt keine ***Sequence Flows*** die Notationselemente innerhalb mit Elementen außerhalb des ***Sub-Process*** verbinden. Ein- und ausgehende ***Sequence Flows*** enden bzw. starten am Rand des ***Sub-Process***.



Sub-Process

Abbildung 4.3.: Graphische Darstellung eines ***Sub-Process***

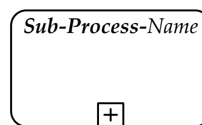
Das einem ***Sub-Process*** zugrunde liegende *BPMN* Diagramm wird im Gegensatz zu einem *UML* Aktivitätsdiagramm weder graphisch abgegrenzt noch graphisch mit einem Namen versehen. Jedes *BPMN* Diagramm wird als separates Modell notiert.

4.3.1.3. *Reusable Sub-Process – Call Task*

Bei ***Reusable Sub-Processes*** spricht die *BPMN* von Platzhaltern für bereits modellierte ***Sub-Processes*** (siehe Unterkapitel 4.3.1.2) oder für solche, deren Verhalten noch nicht feststeht. Da die *BPMN*-Spezifikation zu diesem Aspekt besonders undeutlich ist und andeutet, dass es sich bei ***Reusable Sub-Processes*** lediglich um atomare Tätigkeiten handeln, welche das Verhalten eines ***Sub-Process*** aufrufen und die Kontrolle an diesen übergeben, wird im Rahmen dieser Arbeit in Anlehnung an die aus *UML* Aktivitätsdiagrammen bekannten ***Call Actions*** der Begriff ***Call Task*** geprägt und anstelle von ***Reusable Sub-Process*** eingesetzt (vgl. [OMG09a], S. 59ff).

Ein ***Call Task*** wird wie ein ***Task*** (siehe Unterkapitel 4.3.1.1) über mindestens einen ein- und mindestens einen ausgehenden ***Sequence Flow*** (siehe Unterkapitel 4.3.2.1) in den Ausführungsfluss eingebunden. Über ***Sequence Flows*** eingehende ***Data Objects*** (siehe Unterkapitel 4.3.3) werden durch den ***Call Task*** an den aufgerufenen ***Sub-Process*** zur Bearbeitung übergeben und aus dem ***Sub-Process*** hervorgehende ***Data Objects*** werden vom ***Call Task*** über ausgehende ***Sequence Flows*** weitergereicht.

Die graphische Notation eines ***Call Tasks*** wird nicht eindeutig durch die *BPMN*-Spezifikation vorgegeben. Im einzigen zu diesem Sachverhalt verfügbaren Beispiel werden ***Call Tasks*** wie ***Tasks*** oder ***Sub-Processes*** ohne Inhalt dargestellt, besitzen aber zusätzlich ein Plus-Zeichen in einem Rechteck im unteren Bereich des größeren Rechtsecks (vgl. [OMG09a], S. 59ff). Dieses Zeichen wird aber primär zur Kennzeichnung zugeklappter ***Sub-Processes*** benutzt, d.h. bei ***Sub-Processes*** deren Inhalt im Diagramm nicht dargestellt wird aber bereits modelliert ist (vgl. [OMG09a], S. 56ff). Auf Basis des graphischen Beispiels aus der Spezifikation (vgl. [OMG09a], S. 59, Abb. 9.10) und weil der Bedarf für zugeklappte ***Sub-Processes*** im Rahmen dieser Arbeit nicht besteht, wird die im Beispiel vorgestellte graphische Notation verwendet (siehe Abb. 4.3.1.3).



Call Task

Abbildung 4.4.: Graphische Darstellung eines ***Call Tasks***

4.3.2. *Flow Controls*

Die Sprachelemente in dieser Kategorie regeln den Start, den Ablauf und die Terminierung eines Diagrammes. Dabei wird unterschieden zwischen:

- ***Flows*** (siehe Unterkapitel 4.3.2.1),
- ***Events*** (siehe Unterkapitel 4.3.2.2) und
- ***Gateways*** (siehe Unterkapitel 4.3.2.3).

4.3.2.1. *Flows*

Flows werden als gerichtete Verbindungspfeile zwischen ***Activities*** (siehe Unterkapitel 4.3.1), ***Events*** (siehe Unterkapitel 4.3.2.2) und ***Gateways*** (siehe Unterkapitel 4.3.2.3) verwendet und legen die Ausführungsreihenfolge fest. Zur Beschreibung des Ausführungsflusses sieht die *BPMN*-Spezifikation die Kantenarten ***Sequence Flow*** und ***Message Flow*** vor:

Sequence Flows

Sequence Flows kommen der Bedeutung der ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) aus *UML* Aktivitätsdiagrammen am nächsten. Wie bei der *UML* wird auch in der *BPMN* keine klare Trennung zwischen Kontrollfluss und Datenfluss vorgenommen: ***Sequence Flows*** kombinieren die *UML* ***Activity Edge*** Typen ***Control Flow*** (siehe Unterkapitel 3.3.2.1) und ***Data Flow*** (siehe Unterkapitel 3.3.2.1) in einem Notationselement und eignen sich sowohl zur Lenkung des Ausführungsflusses als auch zum Austausch von ***Data Objects*** (siehe Unterkapitel 4.3.3) zwischen ***Activities***.

Graphisch werden ***Sequence Flows*** als durchgehende Linie mit geschlossener, ausgefüllter Pfeilspitze an der Seite des Zielobjekts dargestellt (siehe Abb. 4.5). Optional möglich sind die Angabe eines Labels und/ oder einer ***Condition***, ein formaler oder informaler Ausdruck, welcher zur Laufzeit ausgewertet wird und als Entscheidungsgrundlage für die Traversierung einer Kante dient. Die *BPMN*-Spezifikation sieht, im Gegensatz zur *UML*, keine Trennung der Darstellung von Label und ***Condition*** vor, stattdessen schlägt sie Werkzeugentwicklern vor, ***Conditions*** als Kantenattribut zu speichern und nicht graphisch darzustellen. **In Anlehnung an die *UML* sollen *Conditions* im Rahmen dieser Arbeit in eckigen Klammern und als mögliche Ergänzung von Labels an *Sequence Flows* geschrieben werden** (siehe Abb. 4.5).

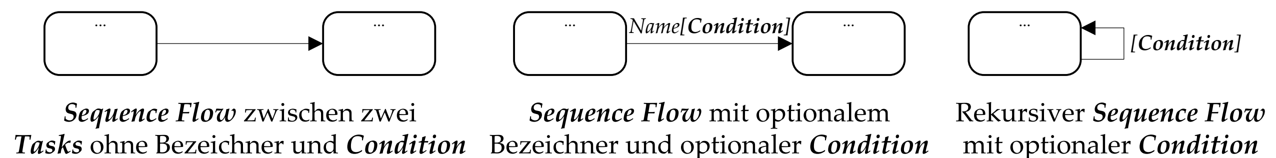
Ein ***Sequence Flow*** hat immer genau eine Quelle und ein Ziel. Ein ***Sequence Flow*** kann dabei auch rekursiv sein. Hierbei ist, sofern nicht beabsichtigt, eine ***Condition*** verpflichtend, da sonst eine Endlosschleife entsteht (siehe Abb. 4.5).

Activities, ***Events*** und ***Gateways*** können, sofern nicht anders angegeben mehrere ein- und ausgehende ***Sequence Flows*** besitzen. **An dieser Stelle unterscheidet sich die *BPMN* von den *UML* Aktivitätsdiagrammen**: mehrere eingehende ***Sequence Flows*** in eine ***Activity***, ein ***Signal Throwing Event*** (siehe Unterkapitel 3.3.2.2), ein ***Signal Catching Event*** (siehe Unterkapitel 3.3.2.2) oder ein ***Final Event*** (siehe Unterkapitel 3.3.2.2) stellen eine implizite ***OR***-Verknüpfung dar, d.h. die

Ausführung des Ziels beginnt sowohl bei Eingang des Ausführungsflusses über einen eingehenden ***Sequence Flow*** als auch beim Eingang mehrerer Ausführungsflüsse über mehrere oder alle eingehenden ***Sequence Flows***. Dieses Konzept ist vergleichbar mit einem zusammenführenden ***Inclusive (Merge) Gateway*** (siehe Unterkapitel 4.3.2.3).

Ein als gesamtes Modell dargestelltes *BPMN* Diagramm besitzt keine ein- und keine ausgehenden ***Sequence Flows***.

Schließlich stellen mehrere von ***Activities, Start Events***, ***Signal Throwing Events*** oder ***Signal Catching Events*** ausgehende ***Sequence Flows*** ein implizites, den Ausführungsfluss aufspaltendes ***AND*** dar, d.h. der Ausführungfluss wird auf mehrere unabhängige, parallel verlaufende Flüsse aufgeteilt. Dieses Konzept ist vergleichbar mit einem den Ausführungsfluss aufspaltenden **Parallel (Branch) Gateway** (siehe Unterkapitel 3.3.2.3).



Sequence Flow zwischen zwei ***Tasks*** ohne Bezeichner und ***Condition***

Sequence Flow mit optionalem Bezeichner und optionaler ***Condition***

Rekursiver ***Sequence Flow*** mit optionaler ***Condition***

Abbildung 4.5.: Graphische Darstellung von ***Sequence Flows***

Message Flows

Message Flows verbinden ausschließlich Notationselemente aus den Zuständigkeitsbereichen verschiedener Akteure (***Participants***) und repräsentieren damit den Nachrichtenaustausch zwischen Akteuren eines Geschäftsprozesses. Genauer genommen verbinden sie Elemente aus verschiedenen ***Pools*** (siehe Unterkapitel 4.3.4) oder die ***Pools*** selbst.

Weil sich ***Message Flows*** nicht wie ***Sequence Flows*** verhalten, z.B. kann ein ***Gateway*** (siehe Unterkapitel 4.3.2.3) keine Quelle und kein Ziel eines ***Message Flows*** sein, und es kein direkt vergleichbares Notationselement in *UML* Aktivitätsdiagrammen gibt, wird **auf die Benutzung von Message Flows im Rahmen dieser Arbeit verzichtet**. Durch die fehlende Verbindungsmöglichkeiten von ***Pools***, sollte auf die Modellierung von mehr als einem ***Pool*** in einem *BPMN* Diagramm verzichtet werden. Stattdessen können die Akteure auch als ***Lanes***, als Untereinheiten eines ***Pools***, dargestellt werden (siehe Unterkapitel 4.3.4).

4.3.2.2. *Events*

Events stellen das Auftreten eines von außerhalb oder innerhalb des *BPMN* Diagrammes eingeleiteten Herganges dar, der sich auf den Ausführungfluss auswirkt, z.Bsp. das Eintreffen eines Signales wie die Kundenanfrage im Einführungsbeispiel (siehe Abb. 4.1).

Die *BPMN*-Spezifikation unterscheidet zwischen ***Events***, welche einen Ausführungsfluss starten (***Start Events***), solchen, die während des Geschäftsprozessen eintreten (***Intermediate Events***) und solchen, die das Ende eines Ausführungsflusses darstellen (***End Event***) (vgl. [OMG09a], S. 35ff). Weiterhin wird zwischen ***Events*** unterschieden, die einen Hergang einleiten/ triggern (***Throwing Events***), und solchen, die einen Hergang abarbeiten (***Catching Events***; vgl. [OMG09a], S. 45).

In die Kategorie der ***Catching Events*** fallen die ***Start Events***. Bei ***Intermediate Events*** kann es sich sowohl um ***Catching Events*** als auch um ***Throwing Events*** handeln. Bei ***End Events*** handelt es sich ausschließlich um ***Throwing Events***. Für jede dieser Kombinationen beschreibt die *BPMN*-Spezifikation verschiedene optionale Auslösertypen, welche die Art des ausgelösten oder bearbeiteten Hergangs beschreiben (vgl. [OMG09a], S. 38ff). So kann ein Event beispielsweise den Empfang oder das Versenden eines Signales, das Auftreten oder die Behandlung eines Fehlers oder die Terminierung des gesamten Geschäftsprozesses darstellen. Jeder Untertyp unterscheidet sich durch eine eindeutige Markierung in der graphischen Repräsentation von den Grund-Symbolen.

In Anlehnung an die Unterkategorien aus dem Unterkapitel zu *UML* ***Event***-Typen in Aktivitätsdiagrammen (siehe Unterkapitel 3.3.2.2) werden die ***Event***-(Unter-)Typen mit den größten semantischen Ähnlichkeiten aus der BPMN-Spezifikation ausgesucht und beschrieben. Unterschieden wird zwischen:

- ***Start Event***, ***End Event*** und ***Terminate End Event*** sowie
- ***Signal Throwing Event*** und ***Signal Catching Event***.

Start Event*, *End Event* und *Terminate End Event

Ein ***Start Event*** ohne Markierung – das Grund-Symbol oder der Obertyp eines ***Start Events*** – legt die Stelle im *BPMN* Diagramm fest, an dem der Ausführungsfluss beginnt und hat damit die gleiche semantische Bedeutung wie ein *UML* ***Initial Node*** (siehe Unterkapitel 3.3.2.2). Mehrere ***Start Events*** in einem *BPMN* Diagramm starten unabhängige, parallel verlaufende Ausführungsflüsse.

Ein Start Event besitzt keinen ein- und mindestens einen ausgehende Sequence Flow. Im Gegensatz zur *UML* dürfen die ausgehenden ***Sequence Flows*** auch Daten transportieren Die ausgehenden ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) eines *UML* ***Initial Nodes*** müssen vom Typ ***Control Flow*** (siehe Unterkapitel 3.3.2.1) sein (vgl. Unterkapitel 3.3.2.2).

Start Event mit optionalem Bezeichner | ***End Event*** mit optionalem Bezeichner | ***Terminate End Event*** mit optionalem Bezeichner

Abbildung 4.6.: Graphische Darstellung eines ***Start Events***, eines ***End Events***, und eines ***Terminate End Events***

Graphisch wird ein ***Start Event*** als leere Kreisfläche mit einfacher Umrandung notiert (siehe Abb. 4.6). Optional lässt sich ein Bezeichner definieren und unter die graphische Repräsentation schreiben.

Das Gegenstück zum ***Start Event*** bilden das ***End Event*** ohne Markierung – das Grund-Symbol oder der Obertyp eines ***End Events*** – und das ***Terminate End Event***, ein Untertyp von ***End Event***. Während ein einfaches ***End Event*** nur die eingehenden Ausführungsflüsse beendet, endet die gesamte Ausführung eines *BPMN* Diagrammes sobald ein Ausführungsfluss ein ***Terminate End Event*** erreicht. ***End Events*** und ***Terminate End Events*** erfüllen die gleiche Rolle wie *UML* ***Flow Final Nodes*** bzw. ***Activity Final Nodes*** (siehe Unterkapitel 3.3.2.2).

Graphisch wird ein ***End Event*** als leere Kreisfläche mit dicker Umrandung dargestellt (siehe Abb. 4.6). Ein ***Terminate End Event*** ergänzt diese Darstellung um einen weiteren, ausgefüllten Kreis im inneren der Kreisfläche (siehe Abb. 4.6). Optional kann ein Bezeichner definiert und unter die graphische Repräsentation notiert werden.

Signal Throwing Event und Signal Catching Event

Bei ***Signal Throwing Events*** und ***Signal Catching Events*** handelt es sich um ***Throwing Events*** und ***Catching Events*** mit einem Auslöser vom Typ ***Signal***. ***Signal Throwing Events*** sind vergleichbar mit ***Send Signal Actions*** aus *UML* Aktivitätsdiagrammen, ***Signal Cacthing Events*** mit ***Accept Signal Actions*** (siehe Unterkapitel 3.3.2.2).

Erreicht der Ausführungsfluss ein ***Signal Throwing Event*** wird ein asynchrones ***Signal*** an ein oder mehrere ***Signal Catching Events*** geschickt, welche nur auf dieses ***Signal*** reagieren. Nach Auslösen des ***Signals*** läuft der Ausführungsfluss am ***Signal Throwing Event*** weiter, parallel werden weitere, unabhängige Ausführungsflüsse an den ***Signal Catching Events*** gestartet. Ein ***Signal Catching Event*** kann ein ***Signal*** auch mehrmals auswerten.

Die Ziele/ Quellen eines ***Signal Throwing Events*** bzw. eines ***Signal Catching Events*** müssen nicht zwingend im gleichen ***Sub-Process*** (siehe Unterkapitel 4.3.1.2) oder dem gleichen *BPMN* Diagramm liegen, sie brauchen auch nicht im aktuell modellierten *BPMN* Diagramm abgebildet sein. Wird ein *BPMN* Diagramm als ***Sub-Process*** in ein weiteres Diagramm eingebettet, sind automatisch alle ***Signal Throwing Events*** und ***Signal Catching Events*** mit gleichem (***Signal***)-Namen miteinander verbunden.

Wenn ein ***Signal Catching Event*** am Anfang des Ausführungsflusses steht, handelt es sich um ein ***Signal Catching Start Event***. Signal Catching Events besitzen keinen ein- dafür aber mindestens einen ausgehenden ***Sequence Flow*** (siehe Unterkapitel 4.3.2.1), welcher auch ***Data Objects*** (siehe Unterkapitel 4.3.3) transportieren kann. Alternativ kann ein ***Signal Catching Event*** im Laufe des Geschäftsprozesses stehen, dann handelt es sich um ein ***Signal Catching Intermediate Event***. Ein ***Signal Catching Intermediate Event*** ist über mindestens einen ein- und einen ausgehenden ***Sequence Flow***, welcher auch ***Data Objects*** transportieren kann, in den Ausführungsfluss eingebunden.

Ein ***Signal Throwing Event*** kann hingegen im Laufe oder am Ende des Geschäftsprozesses auftreten, hierbei handelt es sich um ein ***Signal Throwing Intermediate Event*** bzw. um ein ***Signal Throwing End Event***. Ein ***Signal Throwing Intermediate Event*** hat mindestens einen ein- und mindestens einen ausgehenden ***Sequence Flow***, welcher auch ***Data Objects*** transportieren kann. Ein ***Signal Throwing End Event*** ist über mindestens einen ein- und keinen ausgehenden ***Sequence Flow***, welcher auch ***Data Objects*** transportieren kann, in den Einführungsfluss eingebunden.

Start Events werden als leere Kreisfäche mit einfacher Umrandung, ***Intermediate Events*** als leere Kreisfläche mit doppelter Umrandung und ***End Events*** als leere Kreisfläche mit dicker Umrandung dargestellt (siehe Abb. 4.7). Untertypen wie ***Signal Throwing Events*** und ***Signal Catching Events*** weisen ein Dreickeckssymbol als Unterscheidungsmerkmal in der Kreisinnenfläche auf. Das Symbol eines ***Throwing Events*** wird ausgefüllt, während das eines ***Catching Events*** leer bleibt (siehe Abb. 4.7). Für ***Signal Throwing Events*** und ***Signal Catching Events*** ergeben sich somit insgesamt vier Darstellungsformen, je nachdem an welcher Stelle sie vorkommen und welche Rolle sie im Geschäftsprozess einnehmen (siehe Abb. 4.7).

Signal Catching Start Events sind bei Ausführungsbeginn des Diagrammes bereit Signale zu empfangen und dienen als alternative Startpunkte für die Ausführung. ***Signal Catching Intermediate Events*** sind hingegen erst bereit Signals abzuarbeiten, sobald sie von einem Ausführungsfluss erreicht werden.

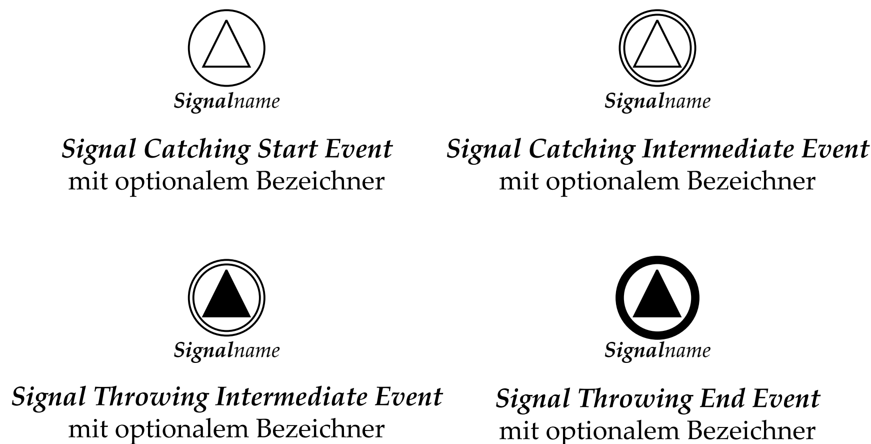


Abbildung 4.7.: Graphische Darstellung von ***Signal Throwing Events*** und ***Signal Catching Events***

4.3.2.3. *Gateways*

Gateways sind für die Verzweigung (***Branch***) und Zusammenführung (***Merge***) von ***Sequence Flows*** (siehe Unterkapitel 3.3.2.1) zuständig und koordinieren den Ausführungsfluss eines Aktivitätsdiagrammes. Obwohl die *BPMN*-Spezifikation die Zusammenlegung aufeinanderfolgender Verzweigungen und Zusammenführungen zu einem einzigen ***Gateway*** erlaubt, wird aus Gründen der

besseren Unterscheidung dieser Notationselemente zwischen ***Branch Gateways*** und ***Merge Gateways*** unterschieden. ***Branch Gateways*** weisen genau einen ein- und mindestens zwei ausgehende ***Sequence Flows*** auf. ***Merge Gateways*** haben mindestens zwei ein- und genau einen ausgehenden ***Sequence Flow***.

In Anlehnung an die ***Control Nodes*** aus den *UML* Aktivitätsdiagrammen (siehe Unterkapitel 3.3.2.3) wird zwischen den folgenden Unterkategorien unterschieden:

- ***Data-based Exclusive Branch Gateways*** und ***Data-based Exclusive Merge Gateways*** sowie
- ***Parallel Branch Gateways*** und ***Parallel Merge Gateways***.

Data-based Exclusive Branch Gateway und Data-based Exclusive Merge Gateway

Exclusive Gateways dienen der Modellierung alternativer Pfade – *XOR*-Verknüpfung – und können als Verzweigung (***Branch***) oder Zusammenführung (***Merge***) verwendet werden (vgl. [All08]). Die *BPMN*-Spezifikation beschreibt zwei Untertypen von ***Exclusive Gateways***:

- ***Data-based Exclusive Branch Gateways*** und ***Data-based Exclusive Mege Gateways*** entsprechen den *UML* ***Decision Nodes*** bzw. den *UML* ***Merge Nodes*** (siehe Unterkapitel 3.3.2.3). Die Entscheidung welcher Pfad eingeschlagen wird, wird, wie bei der *UML,* zur Laufzeit durch Auswertung der Ausdrücke – hier ***Condition*** statt ***Guard*** genannt – an den ausgehenden ***Sequence Flows*** eines ***Data-based Exclusive Branch Gateways*** entschieden. Die Formulierung dieses Ausdrucks kann sowohl informal als auch formal stattfinden. Entgegen der *BPMN*-Spezifikation müssen die **Ausdrücke** an den ausgehenden ***Sequence Flows*** eines ***Data-based Exclusive Branch Gateways*** **im Rahmen dieser Arbeit disjunkt sein** (vgl. [OMG09a], S.75). Der Namenszusatz ***data-based*** kommt zustande weil sich ***Conditions***-Ausdrücke auf Werte von Daten aus dem Geschäftsprozess stützen.
  Graphisch werden ***Data-based Exclusive Branch Gateways*** und ***Data-based Exclusive Merge Gateways*** als leerer Rhombus mit einfacher Umrandung (siehe Abb. 4.8), oder alternativ als Rhombus mit einem X-Symbol im Inneren und einfacher Umrandung dargestellt (siehe Abb. 4.8). Die Angabe eines Bezeichners für ***Data-based Exclusive Branch Gateways*** und ***Data-based Exclusive Merge Gateways*** ist nicht vorgesehen.

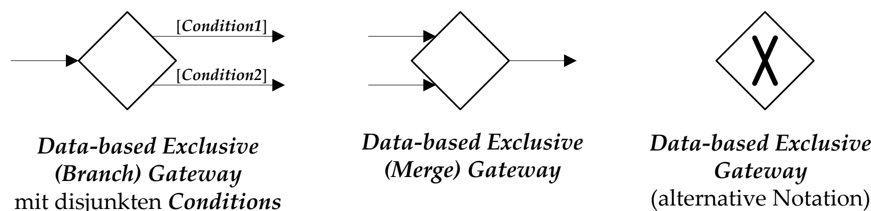


Abbildung 4.8.: Graphische Darstellung von ***Data-based Exclusive Gateways***

- ***Event-based Exclusive Branch Gateways*** und ***Event-based Exclusive Merge Gateways*** haben kein Gegenstück in *UML* Aktivitätsdiagrammen, weshalb sie i**m Rahmen dieser Arbeit**

nur kurz erwähnt aber **nicht verwendet** werden. Die Entscheidung welcher Pfad an einem ***Event-based Exclusive Branch Gateway*** eingeschlagen wird, orientiert sich an Events. Hierzu definiert der Modellierer mehrere exklusive Pfade mit verschiedenen ***Events*** welche an dieser Stelle im Geschäftsprozess auftreten können (vgl. [OMG09a], S. 77ff).

Parallel Branch Gateway und *Parallel Merge Gateway*

Parallel Branch Gateways teilen den Ausführungsfluss in mehrere nebenläufige Flüsse, die unabhängig voneinander abgearbeitet werden und entsprechen damit nahezu *UML* ***Fork Nodes*** (siehe Unterkapitel 3.3.2.3). ***Parallel Branch Gateways*** unterscheiden sich von ***Fork Nodes*** dahingegen, dass ***Conditions*** (siehe ***Data-based Exclusive Branch Gateways***) an ausgehenden Sequence Flows eines Parallel Branch Gateways erlaubt sind, während ***Guards*** für ausgehende ***Activity Edges*** an ***Fork Nodes*** nicht vorgesehen sind. ***Condtitions*** an ausgehenden ***Sequence Flows*** eines ***Parallel Branch Gateways*** sollen den Ausführungsfluss auf einen oder mehrere der möglichen Pfade lenken können. Da dies in *UML* Aktivitätsdiagrammen nicht erlaubt ist und im Rahmen der Modellierung von Geschäftsprozessen in *SOAMIG* nicht benötigt wird, **dürfen ausgehende *Sequence Flows* an *Parallel Branch Gateways* in dieser Arbeit keine *Conditions* haben**.

Dagegen fassen ***Parallel Merge Gateways*** mehrere Ausführungsflüsse zu einem einzigen Fluss zusammen, indem sie den sie verlassenden Ausführungsfluss erst freigeben, nachdem alle in sie eingehende Flüsse an ihr angelangt sind – *AND*-Verknüpfung (vgl. [Kec06]). Sie entsprechen *UML* J***oin Nodes***, besitzen jedoch kein Gegenstück zur optional für einen ***Join Node*** definierbaren ***Join Specification*** (siehe Unterkapitel 3.3.2.3).

Graphisch werden ***Parallel Gateways*** als Rhombus mit einem Pluszeichen im Inneren und einfacher Umrandung dargestellt (siehe Abb.4.9). Die Angabe eines Bezeichners für ***Parallel Branch Gateways*** und ***Parallel Merge Gateways*** ist nicht vorgesehen.

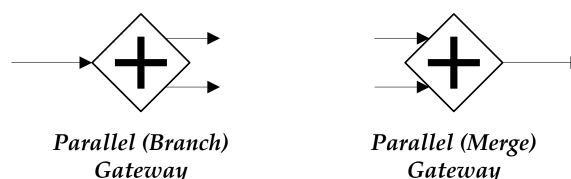


Abbildung 4.9.: Graphische Darstellung von ***Parallel Gateways***

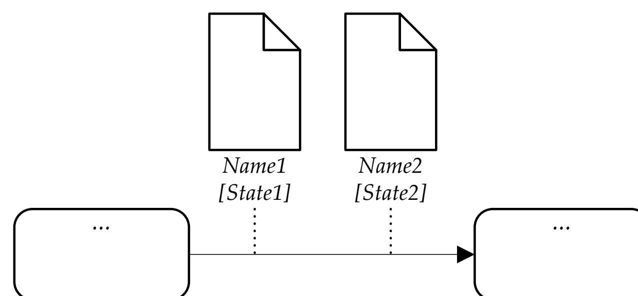
4.3.3. *Resources – Data Objects*

Data Objects stellen die bei der Bearbeitung des Geschäftsprozesses von ***Activities*** (siehe Unterkapitel 4.3.1) verwendeten und produzierte Daten dar. ***Data Objects*** lassen sich einem ***Sub-Process*** (siehe Unterkapitel 4.3.1.2) oder einem *BPMN* Diagramm auch von außerhalb zuführen oder nach

außen abführen. Die ***Object***-Weitergabe zwischen ***Activities*** erfolgt über ***Sequence Flows*** (siehe Unterkapitel 3.3.2.1). Ein ***Sequence Flow*** kann auch mehrere ***Data Objects*** transportieren.

Data Objects weisen syntaktische und semantische Ähnlichkeiten mit *UML* ***Objects*** und *UML* ***Object Nodes*** auf (siehe Unterkapitel 3.3.3). Wie ein *UML* Object repräsentieren ***Data Objects*** Daten im Sinne des objektorientierten Ansatzes. Während ***Objects*** aber nur indirekt über ***Object Nodes*** dargestellt werden, werden ***Data Objects*** als vertikal ausgerichtete Rechtecke mit eingeknickter oberen rechten Ecke notiert (siehe Abb. 4.10). Ähnlichkeiten zu ***Object Nodes*** in Rechtecknotation (siehe Unterkapitel 3.3.3) haben ***Data Objects***, weil sie wie diese graphisch in den ***Object Flow*** bzw. im Fall von ***Data Objects*** über eine gepunktete, ungerichtete Linie an den ***Sequence Flow*** gebunden werden.

Data Objects besitzen einen Bezeichner und einen Zustand, der sich bei der Bearbeitung eines ***Data Objects*** durch eine ***Activity*** ändern kann. Der Bezeichner eines ***Data Objects*** setzt sich im Unterschied zu dem Bezeichner eines *UML* ***Object*** nicht aus einem Instanznamen und einem ***Object***-Typen zusammen, sondern besteht lediglich aus einem allgemeiner gefassten Namen. Bezeichner und Zustand werden untereinander unter der graphischen Repräsentation des ***Data Objects*** notiert, der Zustand wird dabei in eckige Klammer geschrieben (siehe Abb. 4.10).



Zwei Data Objects mit optionaler
Zustandsangabe zwischen zwei Tasks

Abbildung 4.10.: Graphische Darstellung von ***Data Objects***

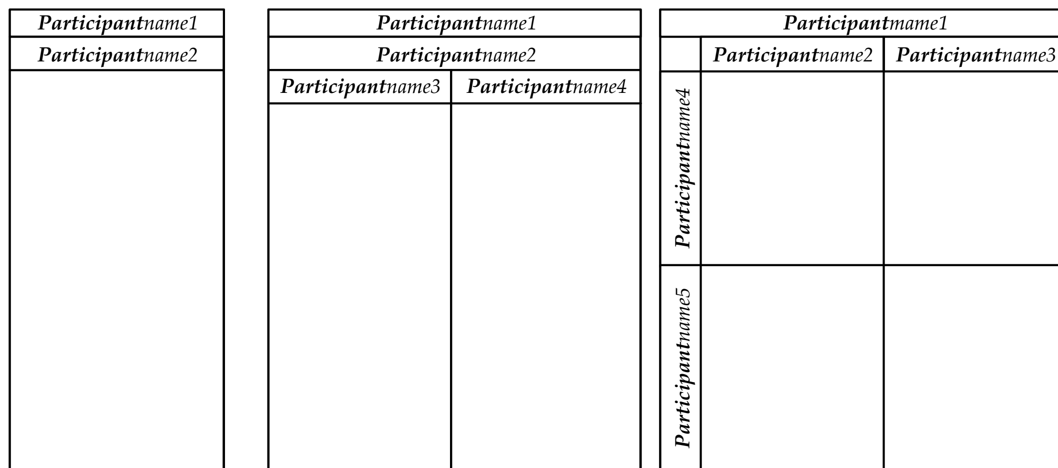
4.3.4. *Participant Partitions – Swimlanes*

Ein Akteur (***Participant***) stellt eine Organisationseinheit in einem Geschäftsprozess dar. Dabei handelt es sich um einen Typ oder eine Rolle, die u.a. eine Firma, ein System, eine Personengruppe oder ein einzelnes Individuum bei der Ausführung des Prozesses einnehmen kann (vgl. [Kec06]). In *BPMN* Aktivitätsdiagrammen wird der Verantwortungsbereich einer Organisationseinheit als ***Swimlane*** modelliert. Alle darin enthaltenen ***Activities*** (siehe Unterkapitel 4.3.1) werden bei der Ausführung des Diagramms durch diesen Akteur ausgeführt.

Pool

Abbildung 4.11.: Graphische Darstellung eines ***Pools***

Die ***Swimlane***-Typen ***Pool*** und ***Lane*** erlauben eine genauere Zuordnung der Aufgaben zu Untereinheiten eines Akteurs. Die Verschachtelung einer oder mehrerer ***Lanes*** in einem ***Pool*** stellt die Aufteilung des Zuständigkeitsbereich eines Akteurs auf mehrere beliebige Untereinheiten dar. Hiermit kann u.a. die interne Struktur einer Firma auf mehrere am Geschäftsprozess mitwirkende Abteilungen abgebildet werden. Im Vergleich zu *UML* Aktivitätsdiagrammen entspricht ein ***Pool*** einer obersten (*top-level*) ***Activity Partition*** (siehe Unterkapitel 3.3.4) eines Aktivitätsdiagrammes, also eine ***Activity Partition*** die in keiner anderen ***Activity Partition*** enthalten ist. Eine ***Lane*** stellt hingegen eine ***Activity Partition*** dar, welche in einer oder mehreren ***Activity Partitions*** enthalten ist. Im Gegensatz zu ***Pools*** dürfen ***Lanes*** nicht direkt in einem *BPMN* Diagramm stehen, sondern müssen direkt oder indirekt über andere ***Lanes*** in einem ***Pool*** liegen.



Pool mit einer ***Lane***

Pool mit mehreren, geschachtelten ***Lanes***

Pool mit gekreuzten ***Lanes***

Abbildung 4.12.: Graphische Darstellung von ***Lanes***

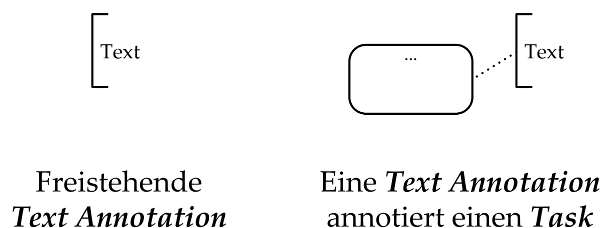
Die *BPMN*-Spezifikation erlaubt Verbindungen zwischen ***Pools*** oder ***Activities*** aus verschiedenen ***Pools*** nur über ***Message Flows***, welche semantisch nicht äquivalent zu ***Sequence Flows*** sind (siehe Unterkapitel 4.3.2.1) und in *UML* Aktivitätsdiagrammen kein Gegenstück besitzen. Durch diesen Umstand und weil sowohl ***Pools*** als auch ***Lanes*** Akteure gleichwärtig darstellen können **wird in dieser Arbeit auf die Verwendung von mehreren *Pools* in einem *BPMN* Diagramm und auf *Message Flows* verzichtet**. Mehrere ***Pools*** werden stattdessen als ***Lanes*** unterhalb eines einzigen ***Pools*** gefasst, welcher auch den Namen des Diagrammes annehmen kann. Dass diese Notation durchaus üblich ist, tritt hervor wenn ein *BPMN* Diagramm keine modellierten ***Pools*** enthält. In diesem Fall enthält das Diagramm nämlich trotzdem einen ***Pool***, welcher aber graphisch nicht dargestellt wird und sich seinen Namen mit dem des Diagrammes teilt – alle Notationselemente befinden sich dann in diesem unsichtbaren ***Pool***.

Graphisch unterscheiden sich ***Pools*** und ***Lanes*** nicht. Sie werden beide als Rechtecke notiert (siehe Abb. 4.11 für ***Pools*** und Abb. 4.12 für ***Lanes***). Je nach Ausrichtung wird der Name des Akteurs in einem weiteren Rechteck im oberen oder im linken Teil notiert. Im Gegensatz zu ***Pools*** lassen sich ***Lanes*** innerhalb von ***Pools*** und anderen ***Lanes*** beliebig tief schachteln (siehe Abb. 4.12) und über Kreuz anordnen (siehe Abb. 4.12).

4.3.5. *Annotations – Text Annotations*

Eine ***Text Annotation*** wird üblicherweise eingesetzt um weitere Informationen über Notationselemente, den Ausführungsfluss, oder Implementierungsdetails anzugeben. Es lässt sich aber prinzipiell jede beliebige Anmerkung (***Annotation***) ausdrücken. ***Text Annotations*** können über eine ungerichtete Verbindung an beliebige Notationselement geheftet werden oder frei stehen, dann gehören sie implizit zu dem Aktivitätsdiagramm/ ***Sub-Process*** (siehe Unterkapitel 4.3.1.2) in dem sie sich befinden. ***Text Annotations*** entsprechen *UML* ***Comments*** (siehe Unterkapitel 3.3.5).

Die Darstellung einer ***Text Annotation*** besteht aus einem offenen Rechteck mit einem beliebigen textuellen Inhalt (siehe Abb.3.14). Ist eine ***Text Annotation*** an ein Element gebunden, wird diese Verbindung graphisch durch eine ungerichtete, gepunktete Linie dargestellt (siehe Abb. 4.13).



Freistehende ***Text Annotation***　　　Eine ***Text Annotation*** annotiert einen ***Task***

Abbildung 4.13.: Graphische Darstellung von ***Text Annotations***

4.4. Fazit

Die *BPMN* bietet umfangreiche Notationsmöglichkeiten für Geschäftsprozesse, welche sich zum Teil direkt und teils durch geschickte Auslegungen der semantisch unpräzisen Spezifikation auf semantisch nahezu äquivalente Gegenstücke aus *UML* Aktivitätsdiagrammen abbilden lassen. Den Anforderungen, für die zu *UML* Aktivitätsdiagrammen semantisch äquivalente Modellierung, von im Rahmen des *SOAMIG*-Projekts erhobenen Geschäftsprozessen, genügen die in diesem Kapitel vorgestellten Sprachkonzepte und Notationselemente. Auf kleinere Diskrepanzen zwischen den beiden Notationen wird im nächsten Teil dieser Arbeit eingegangen und Lösungen vorgestellt (siehe Kapitel 7).

5. *Ereignisgesteuerte Prozessketten* (EPK)

5.1. Einführung in die EPK

Seit Ende der 70er Jahren des letzten Jahrhunderts wurden eine Vielzahl von Modellierungssprachen für Geschäftsprozesse entwickelt. Der Schwerpunkt lag dabei lange Zeit auf der Modellierung der Aufbauorganisation, d.h. der Betrachtung lediglich zeitlich unabhängiger, statischer Regelungen wie Hierarchien und Unternehmenstopologien. Bis Mitte der 90er Jahre des letzten Jahrhunderts gestaltete sich die Koordination über künstlich erzeugte Abteilungsmauern hinweg als schwierig (vgl. [ST05]).

Erst zu dieser Zeit rückte die Ablauforganisation, also das zeitlich-logische, dynamische Verhalten von Vorgängen zur Aufgabenerfüllung in den Vordergrund. Die Kommunikation wurde über Abteilungsgrenzen hinweg erweitert, woraus sich eine Orientierung an der Logik von Geschäftprozessen ergab (vgl. [ST05]).

Für die Konstruktion von Geschäftsprozessmodellen auf fachkonzeptioneller Ebene hat sich aufgrund ihrer Anwendungsorientierung und umfassenden Werkzeuguntestützung, insbesondere im deutschsprachigen Raum, die Ereignisgesteuerte Prozesskette (EPK) etabliert. Sie wurde 1992 in Zusammenarbeit von KELLER, NÜTTGENS und SCHEER am Institut für Wirtschaftsinformatik an der Universität Saarbrücken in Zusammenarbeit mit der *SAP AG* entwickelt. Die EPK fand schnellen Anklang und wurde zu einem zentralen Bestandteil der *SAP*-Referenzmodelle und der *ARIS*-Konzepte und somit Grundlage modellgetriebener Ansätze für ein durchgängiges und werkzeuggestütztes Geschäftsprozessmanagement (vgl. [NR02]).

EPK stellen eine semi-formale, graphische Methode zur Modellierung von Geschäftsprozessmodellen als bipartite Graphen aus Funktionen und Ereignissen dar. Semantik und Syntax des Sprachkerns wurden in mehreren Arbeiten formal beschrieben. Zusätzlich wurde die Sprache um eine Vielzahl von Konzepten erweitert. In diesem Zusammenhang wird in der Literatur oft von erweiterten Ereignisgesteurerten Prozessketten (eEPK) gesprochen.

Als Ausgangspunkt für die Wahl zu *UML* Aktivitätsdiagrammen und *BPMN* kompatibler Sprachkonzepte und Notationselemente dienen einerseits die bereits 1992 in der Urfassung der Sprache festgelegten Konzepte sowie erweiterte, auf dem *ARIS*-Sichtkonzept aufbauende Konzepte, wie die Zuordnung von Tätigkeiten zu Geschäftsprozessakteuren oder die Repräsentation von Ressourcen.

5.2. Notationsübersicht

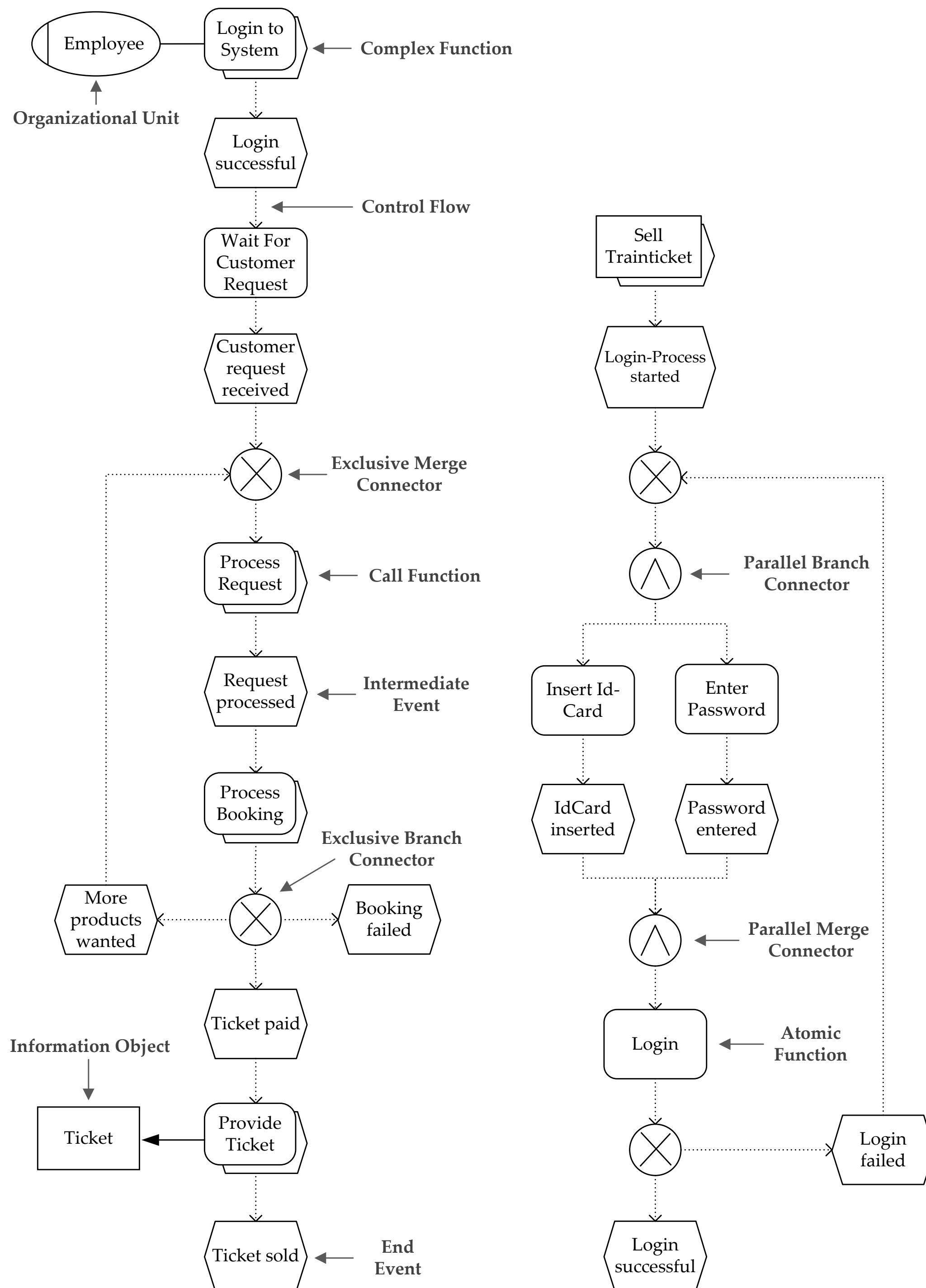


Abbildung 5.1.: Modellierung des Geschäftsprozesses *sellTrainticket* als EPK

Zur Einführung in die EPK betrachten wir in Abbildung 5.1 die Modellierung des Geschäftsprozesses *sellTrainticket*, welcher bereits bei der Notationsübersicht der *UML* Aktivitätsdiagramme und der *BPMN* beschrieben wurde (siehe Unterkapitel 3.2 und 4.2). Unterschiede in der graphischen Repräsentation ergeben sich durch den Einsatz unterschiedlicher Notationen, doch die Diagramme lassen sich sehr ähnlich lesen und mit dem Verständnis der jeweils anderen Darstellungen nachvollziehen.

Die Semantik des Geschäftsprozesses wurde bei der Abbildung in die EPK erhalten: ein Angestellter eines Reisebürounternehmens meldet sich mit Passwort und Identifikationskarte an seinem Computersystem an und ist dann bereit Kundenanträge zu bearbeiten, zu buchen und den Kunden die bezahlten Fahrkarten auszuhändigen.

Als auffälligster Unterschied fällt die Aufteilung des Geschäftsprozessmodells auf zwei EPK auf: in der Abbildung wird links der Hauptprozess *Sell Trainticket* und rechts der Unterprozess *Login to System* beschrieben. Während die *UML* Aktivitätsdiagramme und die *BPMN* dem Modellierer mit ***Activities*** (siehe Unterkapitel 3.3.1.2) bzw. ***Sub-Processes*** (siehe Unterkapitel 4.3.1.2) die Möglichkeit geben zusammengesetzte Tätigkeiten direkt innerhalb der graphischen Repräsentation des Notationselementes zu notieren, muss der Modellierer bei EPK für jede zusammengesetzte Tätigkeit (***Complex Function***, siehe Unterkapitel 5.3.1.2) eine separate EPK erstellen.

Im Unterschied zu einem *UML* Aktivitätsdiagramm wird eine EPK wie ein *BPMN* Diagramm graphisch nur durch die enthaltenen Notationselemente dargestellt. Auch die Modellierung von Geschäftsprozessakteuren (im Beispiel *Employee*) als ***Organizational Unit*** (siehe Unterkapitel 5.3.4) unterscheidet sich von der Repräsentation als ***Activity Partition*** (siehe Unterkapitel 3.3.4) aus *UML* Aktivitätsdiagrammen und ***Pools*** (siehe Unterkapitel 4.3.4) aus *BPMN* Diagrammen. Anstatt den Verantwortungsbereich graphisch als ein Behältnis, in dem alle, durch den zuständigen Akteur auszuführende Tätigkeiten stehen, darzustellen, wird eine Tätigkeit in EPK einem Akteur über eine ungerichtete Kante zugeteilt. Sofern nicht durch weitere/ andere ***Organizational Units*** angegeben, werden alle im Kontrollfluss folgenden Tätigkeiten von dieser ***Organizational Unit*** ausgeführt.

Ein weiteres auffälliges Merkmal von EPK ist die bipartite Darstellung des Graphens. Gerichtete Kanten (***Control Flow***, siehe Unterkapitel 5.3.2.1) steuern den Ausführungsfluss durch alternierendes verbinden von Tätigkeiten (***Functions***, siehe Unterkapitel 5.3.1) und Ereignissen (***Events***, siehe Unterkapitel 5.3.2.2), auch über zwischengeschaltete Verknüpfungen (***Connectors***, siehe Unterkapitel 5.3.2.3).

Anmerkungen und Kommentare (***Descriptions***, siehe Unterkapitel 5.3.5) zum Geschäftsprozess oder zum Ablauf können, im Gegensatz zu *UML* ***Comments*** (siehe Unterkapitel 3.3.5) und *BPMN* ***Text Annotations*** (siehe Unterkapitel 4.3.5), nicht graphisch dargestellt werden. Einen weiteren Unterschied gibt es beim Austausch von Daten zwischen ***Functions***. Während in *UML* Aktivitätsdiagrammen ***Objects*** direkt an ***Object Flows*** (siehe Unterkapitel 3.3.3 und 3.3.2.1) und in *BPMN* Diagrammen ***Data Objects*** direkt an ***Sequence Flows*** (siehe Unterkapitel 4.3.3 und 4.3.2.1) gebunden werden und damit den Transport der Datensätze durch diese Kanten darstellen, werden ***Information Objects*** (siehe Unterkapitel 5.3.3) in EPK direkt mit ***Functions*** verbunden. Ein- und Ausgabedaten unterscheiden sich durch die Richtung ihrer Verbindungen zu einer ***Function***, ein

Pfeil in Richtung einer ***Function*** beschreibt ein Eingabedatum, ein Pfeil in Richtung eines ***Information Objects*** beschreibt hingegen ein Ausgabedatum.

Trotz aller beschriebener Unterschiede gibt es auch Übereinstimmungen zwischen den Sprachen: ***Functions*** stellen Konzepte zur Beschreibung von Tätigkeiten zur Verfügung und entsprechen weitgehend *UML* ***Activity Nodes*** (siehe Unterkapitel 3.3.1) und *BPMN* ***Activities*** (siehe Unterkapitel 4.3.1). ***Atomic Functions*** beschreiben atomare Tätigkeiten wie die Passworteingabe im Beispiel (*Enter Password*) und entsprechen *UML* ***Actions*** (siehe Unterkapitel 3.3.1.1) sowie *BPMN* ***Tasks*** (siehe Unterkapitel 4.3.1.1). ***Complex Functions*** beschreiben zusammengesetzte Tätigkeiten wie den Unterprozess *Login To System* und haben große Ähnlichkeiten mit *UML* ***Activities*** (siehe Unterkapitel 3.3.1.2) und *BPMN* ***Sub-Processes*** (siehe Unterkapitel 4.3.1.2) aber auch, durch die fehlende graphische Darstellung des Inhalts im Notationselement, mit *UML* ***Call Actions*** (siehe Unterkapitel 3.3.1.3) und *BPMN* ***Call Tasks*** (siehe Unterkapitel 4.3.1.3)(vgl. ***Call Function*** im Unterkapitel 5.3.1.3).

Ähnlichkeiten zwischen den Notationen gibt es auch bei den Ereignissen (***Events***, siehe Unterkapitel 5.3.2.2): ***Start Events*** und ***End Events*** starten und beenden den Ausführungsfluss eines Geschäftsprozess und entsprechen *UML* ***Initial Nodes*** und ***Final Nodes***, sowie *BPMN* ***Start Events*** und ***End Events***. Speziell gekennzeichnete ***Events*** als Gegenstück zu *UML* ***Send Signal Actions*** und ***Accept Signal Actions*** (siehe Unterkapitel 3.3.2.2) bzw. *BPMN* ***Signal Throwing Events*** und ***Signal Catching Events*** (siehe Unterkapitel 4.3.2.2) gibt es in der EPK hingegen nicht (siehe Darstellung von *Customer Request Received* in Abb. 5.1 im Vergleich zur Darstellung von *receiveCustomerRequest* in Abb. 3.1 und Abb. 4.1).

Weitere Übereinstimmungen gibt es bei der Aufteilung (***Branch***) und Zusammenführung (***Merge***) von Ausführungsflüssen durch ***Connectors*** (siehe Unterkapitel 5.3.2.3), welche in *UML* Aktivitätsdiagrammen unter dem Begriff ***Control Nodes*** (siehe Unterkapitel 3.3.2.3) und in *BPMN* unter dem Begriff ***Gateways*** (siehe Unterkapitel 4.3.2.3) wiederzufinden sind. Hiermit lassen sich mehrere Ausführungsflüsse bündeln bevor bzw. aufteilen nachdem sie eine ***Function*** oder ein ***Event*** erreichen bzw. verlassen.

5.3. Notationselemente

Im folgenden werden die Sprachkonzepte und Notationselemente der EPK beschrieben, die sich in die in der Einleitung zu diesem Kapitel (siehe Kapitel 2) identifizierten Elementgruppen einordnen lassen und für die Erhebung von Geschäftsprozessmodellen im *SOAMIG*-Projekt benötigt werden. **Um eine möglichst große Übereinstimmung bei der Modellierungsmöglichkeiten zu erhalten, wurden die Elemente in Anlehnung an Sprachelemente aus den *UML* Aktivitätsdiagrammen und den *BPMN* Diagrammen ausgewählt.** Im Laufe der Beschreibungen wird auf die jeweiligen Gegenstück aus den anderen Notationen hingewiesen. Die Elemente sind den folgenden fünf Kategorien zugeordnet:

- ***Activities – Functions*** (siehe Unterkapitel 5.3.1),
- ***Flow Controls*** (siehe Unterkapitel 5.3.2),
- ***Resources – Information Objects*** (siehe Unterkapitel 5.3.3),
- ***Participant Partitions – Organizational Units*** (siehe Unterkapitel 5.3.4) und
- ***Annotations – Descriptions*** (siehe Unterkapitel 5.3.5).

5.3.1. *Activities – Functions*

Diese Kategorie beinhaltet Sprachelemente zur Beschreibung des Verhaltens einer EPK und bildet einfache und komplexe Arbeitsabläufe eines Geschäftsprozesses ab. Es wird unterschieden zwischen:

- ***Atomic Functions*** (siehe Unterkapitel 5.3.1.1),
- ***Complex Functions*** (siehe Unterkapitel 5.3.1.2) und
- ***Call Functions*** (siehe Unterkapitel 5.3.1.3).

5.3.1.1. *Atomic Function*

Eine ***Atomic Function*** beschreibt einen elementaren, nicht weiter zerlegbaren oder nicht näher dargestellten Arbeitsablauf eines Prozesses und entspricht einer ***Action*** aus *UML* Aktivitätsdiagrammen (siehe Unterkapitel 3.3.1.1) und einem ***Task*** aus BPMN Diagrammen (siehe Unterkapitel 4.3.1.1).

Eine ***Atomic Function*** wird über genau einen ein- und einen ausgehenden ***Control Flow*** (siehe Unterkapitel 5.3.2.1) in den Ausführungsfluss eingebunden, d.h. über gerichtete Verbindungen mit anderen Notationselementen verbunden. Auf eine ***Atomic Function*** folgt ein ***Event*** (siehe Unterkapitel 5.3.2.2), auch über ***Connectors*** (siehe Unterkapitel 5.3.2.3) hinweg, welches das Ergebnis/ die Ergebnisse der Tätigkeit beschreiben. Über ***Information Objects*** (siehe Unterkapitel 5.3.3) können einer ***Atomic Function*** Daten zur Bearbeitung zugeführt und durch ***Atomic Function*** produzierte Datensätze an weitere ***Functions*** weitergegeben werden. Eine ***Atomic Function*** lässt sich über ***Organization Units*** (siehe Unterkapitel 5.3.4) Geschäftsprozessakteuren zur Ausführung zuweisen.

Die graphische Notation besteht aus einem Rechteck mit abgerundeteten Ecken und einer beliebig detaillierten formalen oder informalen Beschriftung, welche an beliebiger Stelle in oder um das Rechteck positioniert wird (siehe Abb. 5.2).

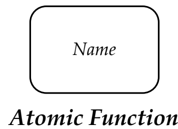


Abbildung 5.2.: Graphische Darstellung einer ***Atomic Function***

5.3.1.2. *Complex Function*

Eine zusammengesetzte Tätigkeit lässt sich über eine ***Complex Function*** in einer weiteren *EPK* beschreiben und ähnelt ***Activities*** aus UML Aktivitätsdiagrammen (siehe Unterkapitel 3.3.1.2) sowie ***Sub-Processes*** aus *BPMN* Diagrammen (siehe Unterkapitel 4.3.1.2). ***Complex Functions*** werden wie ***Atomic Functions*** (siehe Unterkapitel 5.3.1.1) über ein- und ausgehende ***Control Flows*** (siehe Unterkapitel 5.3.2.1) in den Ausführungsfluss integriert und können ***Information Objects*** (siehe Unterkapitel 5.3.3) erhalten, bearbeiten und ausgeben, sowie einer ***Organizational Unit*** (siehe Unterkapitel 5.3.4) zur Ausführung zugeteilt werden.

Eine ***Complex Function*** wird, sofern sie nicht am Anfang oder am Ende des Ausführungsflusses steht, von mindestens einem ***Event*** (siehe Unterkapitel 5.3.2.2) vorangegangen und mindestens einem ***Event*** nachgegangen. Diese ***Events*** stellen die Vor- und Nachzustände der ***Complex Function*** dar und haben die gleichen Namen wie die ***Start Events*** und ***End Events*** des Unterprozesses (siehe Abb. 5.3).

In der Literatur wird eine ***Complex Function*** graphisch unterschiedlich dargestellt. In dieser Arbeit wird eine ***Complex Function*** als Rechteck mit abgerundeten Ecken vor einem Sechseck notiert (siehe Abb. 5.3). Der Bezeichner wird an beliebiger Stelle in oder um die graphische Darstellung notiert und entspricht dem Namen der EPK/ des Unterprozesses welche(r) der ***Complex Function*** zugrunde liegt. Im Gegensatz zu ***Activities*** aus der *UML* sowie ***Sub-Processes*** aus der *BPMN* wird der Inhalt einer ***Complex Function*** nicht im Symbol angezeigt.

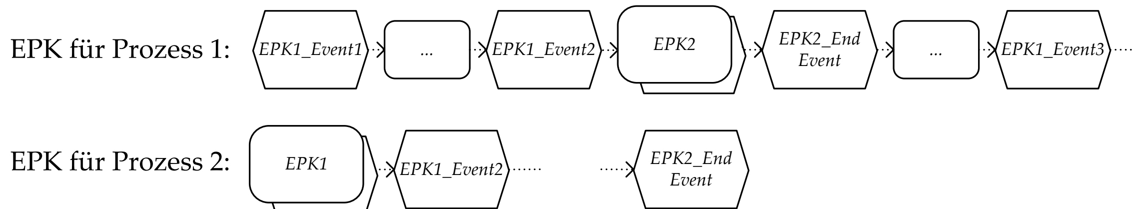


Abbildung 5.3.: Graphische Darstellung einer ***Complex Function***

5.3.1.3. *Call Function*

Eine ***Call Function***, analog zu *UML* ***Call Actions*** (siehe Unterkapitel 3.3.1.3) und *BPMN* ***Call Tasks*** (siehe Unterkapitel 4.3.1.3) ist eigentlich in der EPK-Notation nicht vorgesehen. Der Inhalt der EPK für die dieses Element als Platzhalter fungiert muss eigentlich bereits feststehen, womit das Element einer reinen ***Complex Function*** entsprechen würde (siehe Unterkapitel 5.3.1.2). Die ***Events*** (siehe Unterkapitel 5.3.2.2) vor und nach einer ***Complex Function***, die auf die ***Start Events*** und ***End Events*** des dargestellten Unterprozesses abgebildet werden, lassen sich nach dem *Black Box*-Konzept aus der Systemtheorie als Schnittstellen für die durch den Unterprozess beschriebene Funktionalität verstehen. Da diese Schnittstellen/ Vor- und Nachzustände auch für die Ausführung einer zusammengesetzten aber nicht innerlich modellierten Tätigkeit bereits feststehen sollten, bevor der Unterprozess in einen Geschäftsprozess eingesetzt wird, wird im Rahmen dieser Arbeit der Begriff ***Call Function*** in Anlehnung an die *UML* ***Call Actions*** und *BPMN* ***Call Tasks*** geprägt. Um die Analogie zu UML ***Call Actions*** und *BPMN* ***Call Tasks*** zu gewährleisten wird festgelegt, dass eine ***Call Function*** bei Eintreffen des Ausführungsflusses lediglich den zugrundeliegenden Unterprozess/ EPK aufruft und die Kontrolle an dieses weitergibt, womit es sich bei ***Call Functions*** um eine atomare Tätigkeit handelt.

Analog zu einer ***Complex Function*** soll eine ***Call Function***, sofern sie nicht am Anfang oder Ende des Ausführungsflusses vorkommt, über einen ein- und einen ausgehenden ***Control Flow*** (siehe Unterkapitel 5.3.2.1) verfügen und kann ***Information Objects*** (siehe Unterkapitel 5.3.3) erhalten, bearbeiten und ausgeben, sowie einer ***Organizational Unit*** (siehe Unterkapitel 5.3.4) zur Ausführung zugeteilt werden. Die ***Events*** vor und nach einer ***Call Function*** stellen die ***Start Events*** und ***End Events*** des Unterprozesses, dessen Verhalten aufgerufen werden soll, dar.

Graphisch wird eine ***Call Function*** wie eine ***Complex Function*** dargestellt (siehe Abb. 5.4). Der Bezeichner wird an beliebiger Stelle in oder um die graphische Darstellung notiert und entspricht dem Namen des Unterprozesses/ der EPK dessen Verhalten aufgerufen werden soll.

Call Function

Abbildung 5.4.: Graphische Darstellung einer ***Call Function***

5.3.2. *Flow Controls*

Die Sprachelemente in dieser Kategorie regeln den Start, den Ablauf und die Terminierung eines Diagrammes. Dabei wird unterschieden zwischen:

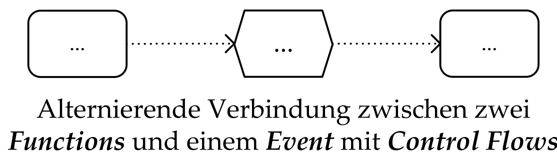
- ***Control Flows*** (siehe Unterkapitel 5.3.2.1),
- ***Events*** (siehe Unterkapitel 5.3.2.2) und
- ***Connectors*** (siehe Unterkapitel 5.3.2.3).

5.3.2.1. *Control Flow*

Control Flows stellen gerichtete Verbindungen zwischen ***Functions***, ***Events*** und ***Connectors*** (siehe Unterkapitel 5.3.1, 5.3.2.2 und 5.3.2.3) dar und legen die Ausführungsreihenfolge einer EPK fest. Ein **Control Flow** hat immer genau eine Quelle und ein Ziel. Da sich ***Functions*** und ***Events*** im Ausführungsfluss, auch über ***Connectors*** hinweg, abwechseln müssen (vgl. [GK92]), sind Rekursionen und das direkte Verbinden zweier ***Functions*** oder ***Events*** nicht erlaubt (siehe Abb. 5.5). ***Control Flows*** lassen sich mit *UML* ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) und *BPMN* ***Sequence Flows*** (siehe Unterkapitel 4.3.2.1) vergleichen. ***Control Flows*** transportieren im Gegensatz zu ***Sequence Flows*** aber keine Daten (siehe Unterkapitel 5.3.3).

Eine EPK besitzt keine ein- und ausgehenden ***Control Flows***, ***Functions*** besitzen genau einen ein- und einen ausgehenden ***Control Flow*** und ***Events*** mindestens einen ein- oder einen ausgehenden ***Control Flow***.

Graphisch präsentiert sich ein ***Control Flow*** als gepunktete Linie mit offener Pfeilspitze an der Seite des Zielobjekts (siehe Abb. 5.5). Die Angabe eines Bezeichners und einer Ausführungsbedingung (***Guards***/ ***Condition***) sind nicht vorgesehen.



Alternierende Verbindung zwischen zwei
Functions und einem ***Event*** mit ***Control Flows***

Abbildung 5.5.: Graphische Darstellung von ***Control Flows***

5.3.2.2. *Events*

Events stellen das Auftreten eines, von innerhalb oder außerhalb einer *EPK* eingeleiteten, Herganges dar, der sich auf den Ausführungfluss auswirkt, z.B. das Eintreffen einer Kundenanfrage im Einführungsbeispiel (siehe Abb. 5.1).

Die EPK unterscheidet zwischen ***Events*** welche einen Ausführungsfluss starten (***Start Events***), solchen die während eines Geschäftsprozesses eintreten (***Intermediate Events***) und solchen die das Ende eines Ausführungsflusses darstellen (***End Events***). Weitere Untertypen, wie ***Signal Events*** aus der *UML* und der *BPMN* (siehe Unterkapitel 3.3.2.2 und Unterkapitel 4.3.2.2), sind nicht vorgesehen. Ein ***Start Event***, ein ***Intermediate Event*** oder ein ***End Event*** kann aber je nach Auslegung als ***Signal Event*** interpretiert werden.

Bei einem ***Start Event*** handelt es sich um den Auslöser der Prozessausführung. Ein ***Start Event*** besitzt keinen ein- und genau einen ausgehenden ***Control Flow*** (siehe Abb. 5.6) und entspricht einem *UML* ***Initial Node*** (siehe Unterkapitel 3.3.2.2) sowie einem *BPMN* ***Start Event*** (siehe Unterkapitel 4.3.2.2).

Ein ***Intermediate Event*** tritt im Laufe des Ausführungsflusses auf und nimmt verschiedene Funktionen ein. Primär lassen sich ***Intermediate Events*** als Vor- oder Nachzustände von ***Functions*** verstehen, alternativ können sie, analog zu *UML* ***Guards*** an ***Activity Edges*** (siehe Unterkapitel 3.3.2.1) und *BPMN* ***Conditions*** an ***Sequence Flows*** (siehe Unterkapitel 4.3.2.1), als Entscheidungsgrundlagen an ausgehenden ***Control Flows*** von ***Exclusive Branch Connectors***, ***Inclusive Branch Connectors*** (siehe Unterkapitel 5.3.2.3) oder ***Functions*** eingesetzt werden. Ein ***Intermediate Event*** unterscheidet sich nur dahingegen von einem ***Start Event*** und einem ***End Event***, dass es sowohl einen ein- wie auch einen ausgehenden ***Control Flow*** besitzt (siehe Abb. 5.6).

Erreicht der Ausführungsfluss ein ***End Event***, stoppt er an dieser Stelle. Bei parallelen Vorgängen, eingeleitet durch einen ***Parallel Branch Connector*** (siehe Unterkapitel 5.3.2.3), terminiert nur der Ausführungszweig welcher ein ***End Event*** erreicht. Nur wenn alle Ausführungszweige auf ***End Events*** stoßen, terminiert die gesamte Prozessausführung. Ein ***End Event*** besitzt genau eine ein- und keine ausgehende Kontrollflusskanten (siehe Abb. 5.6). Eine Unterscheidung zwischen ***End Events*** welche analog zu *UML* ***Activity Final Nodes*** (siehe Unterkapitel 3.3.2.2) und *BPMN* ***Terminate End Events*** (siehe Unterkapitel 4.3.2.2) unabhängig davon ob der Ausführungsfluss auf mehrere Zweige aufgeteilt wurde oder nicht, den gesamten Prozess terminieren und solchen die analog zu *UML* ***Flow Final Nodes*** (siehe Unterkapitel 3.3.2.2) und *BPMN* ***End Events*** (siehe Unterkapitel 4.3.2.2) nur für die Terminierung des einlaufenden Ausführungszweiges zuständig sind, lässt sich vom Modellierer nicht explizit definieren. Ein ***End Event*** unterscheidet sich nur dahingegen von einem ***Start Event*** und einem ***Intermediate Event*** dass er keinen ein- dafür genau einen ausgehenden ***Control Flow*** besitzt (siehe Abb. 5.6).

Start Events, ***Intermediate Events*** und ***End Events*** werden als Sechseck mit einer Beschriftung in oder um das Sechseck dargestellt (siehe Abb. 5.6). Bei ***Events*** handelt es sich um passive Komponenten, weshalb sie keinem Akteur/ keiner ***Organizational Unit*** (siehe Unterkapitel 5.3.4) zugeordnet werden.

Start Event, ***Intermediate Event*** und ***End Event***
jeweils mit optionalem Bezeichner

Abbildung 5.6.: Graphische Darstellung eines ***Start Events***, ***Intermediate Events*** und ***End Events***

5.3.2.3. ***Connectors***

Connectors sind für die Verzweigung (***Split*** oder ***Branch***) und Zusammenführung (***Join*** oder ***Merge***) von ***Control Flows*** (siehe Unterkapitel 5.3.2.1) zuständig und koordinieren den Ausführungsfluss einer EPK. Obwohl die EPK-Notation die Zusammenlegung aufeinanderfolgender Verzweigungen und Zusammenführungen zu einem einzigen Symbol erlaubt, wird aus Gründen der besseren Unterscheidung der Notationselemente, zwischen ***Branch Connectors*** und ***Merge Connectors*** unterschieden. ***Branch Connectors*** weisen genau einen ein- und mindestens zwei ausgehende ***Control Flows*** auf. ***Merge Controls*** haben mindestens zwei ein- und genau einen ausgehenden ***Control Flow***.

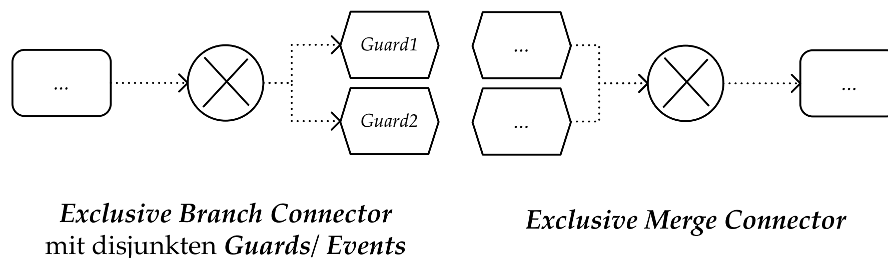
In Anlehnung an die ***Control Nodes*** aus *UML* Aktivitätsdiagrammen (siehe Unterkapitel 3.3.2.3) und ***Gateways*** aus *BPMN* Diagrammen (siehe Unterkapitel 4.3.2.3) wird zwischen den folgenden Unterkategorien unterschieden:

- ***Exclusive Branch Connectors*** und ***Exclusive Merge Connectors***,
- ***Parallel Branch Connectors*** und ***Parallel Merge Connectors*** sowie
- ***Inclusive Branch Connectors*** und ***Inclusive Merge Connectors***.

Exclusive Branch Connector und ***Exclusive Merge Connector***

Ein disjunktiver Verknüpfungsoperator – logischer *XOR*-Operator – lenkt den Ausführungsfluss auf einen der möglichen Pfade (***Exclusive Branch Connector***) oder fasst mehrere disjunkte Pfade zusammen (***Exclusive Merge Connector***). Die Entscheidungsgrundlage für die Wahl eines Pfades beruht auf dem Ergebnis einer dem Verknüpfungsoperator vorangestellten ***Function*** (siehe Unterkapitel 5.3.1). Die möglichen Ergebnisse und damit die möglichen Pfade sind durch dem Verknüpfungsoperator nachgestellte ***Events*** (siehe Unterkapitel 5.3.2.2) modelliert (vergleichbar mit ***Guards*** an ***Activity Edges*** in *UML* oder ***Conditions*** an ***Sequence Flows*** in *BPMN*, siehe Unterkapitel 3.3.2.1 bzw. Unterkapitel 4.3.2.1).

Graphisch werden ***Exclusive Branch Connectors*** und ***Exclusive Merge Connectors*** als Kreisfläche mit einfacher Umrandung und X-Symbol im Inneren dargestellt (siehe Abb. 5.7).



Exclusive Branch Connector
mit disjunkten ***Guards***/ ***Events***

Exclusive Merge Connector

Abbildung 5.7.: Graphische Darstellung eines ***Exclusive Branch Connectors*** und eines ***Exclusive Merge Connectors***

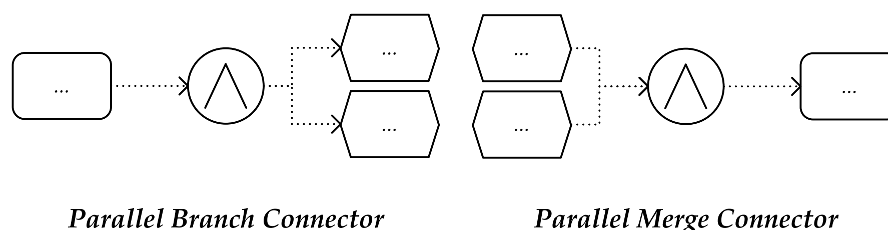
Exclusive Branch Connectors und ***Exclusive Merge Connectors*** sind mit *UML* ***Decision Nodes*** respektiv ***Merge Nodes*** (siehe Unterkapitel 3.3.2.3) beziehungsweise mit *BPMN* ***Data-based Exclusive Branch Gateways*** respektiv ***Data-based Exclusive Merge Gateways*** (siehe Unterkapitel 4.3.2.3) vergleichbar.

Parallel Branch Connector und *Parallel Merge Connector*

Ein konjunktiver Verknüpfungoperator – logischer *AND*-Operator – teilt den Kontrollfluss auf mehrere, unabhängige, parallel verlaufende Ausführungsflüsse (***Parallel Branch Connector***) oder führt mehrere parallele Flüsse zusammen (***Parallel Merge Connector***). Der Kontrollfluss an einem ***Parallel Branch Connector*** läuft nur dann weiter wenn alle Fälle einer Entscheidungsgrundlage eintreten oder keine Entscheidung gefällt wird. Ersteres wird durch nachstehende ***Events*** (siehe Unterkapitel 5.3.2.2) und eine vorangestellte ***Function*** (siehe Unterkapitel 5.3.1) modelliert, während zweiteres durch ein vorangehendes ***Event***, welches als passives Element in einer *EPK* keine Entscheidungskompetenz besitzt, dargestellt wird. An einem ***Parallel Merge Connector*** müssen erst alle Kontrollflüsse einlaufen, damit der Kontrollfluss zusammengeführt wird und weiterlaufen kann.

Graphisch werden ***Exclusive Branch Connectors*** und ***Exclusive Merge Connectors*** als Kreisfläche mit einfacher Umrandung und einem logischen *AND*-Symbol im Inneren dargestellt (siehe Abb. 5.8).

Parallel Branch Connectors und ***Parallel Merge Connectors*** sind mit *UML* ***Fork Nodes*** respektiv ***Join Nodes*** (siehe Unterkapitel 3.3.2.3) beziehungsweise mit *BPMN* ***Parallel Branch Gateways*** respektiv ***Parallel Merge Gateways*** (siehe Unterkapitel 4.3.2.3) vergleichbar.



Parallel Branch Connector ***Parallel Merge Connector***

Abbildung 5.8.: Graphische Darstellung eines ***Parallel Branch Connectors*** und eines ***Parallel Merge Connectors***

Inclusive Branch Connector und *Inclusive Merge Connector*

Ein adjunktiver Verknüpfungsoperator – logischer *OR*-Operator – lenkt den Kontrollfluss auf einen oder mehrere mögliche Pfade (***Inclusive Branch Connector***) oder fasst mehrere Pfade zusammen (***Inclusive Merge Connector***). Die Entscheidungsgrundlage für die Wahl eines Pfades beruht

auf dem Ergebnis einer dem Verknüpfungsoperator vorangestellten ***Function*** (siehe Unterkapitel 5.3.1). Die möglichen Ergebnisse und damit die möglichen Pfade sind durch dem Verknüpfungsoperator nachgestellte ***Events*** modelliert (vergleichbar mit ***Guards*** an ***Activity Edges*** in *UML* (siehe Unterkapitel 3.3.2.1) oder ***Conditions*** an ***Sequence Flows*** in *BPMN* (siehe Unterkapitel 4.3.2.1).

Während ***Inclusive Branch Connectors*** in dieser Arbeit nicht benötigt werden, werden ***Inclusive Merge Connectors*** u.a. gebraucht um mehrere ***Control Flows*** (siehe Unterkapitel 5.3.2.1) zusammenzuführen, welche als Abbilder von ***Sequence Flows*** in einem *BPMN* Diagramm in eine ***Activity*** (siehe Unterkapitel 4.3.1) fließen, bevor sie in das ***Function***-Abbild der ***Activity*** laufen.

Graphisch werden ***Inclusive Branch Connectors*** und ***Inclusive Merge Connectors*** als Kreisfläche mit einfacher Umrandung und einem logischen *OR*-Symbol im Inneren dargestellt (siehe Abb. 5.9).

Für ***Inclusive Merge Connectors*** wird in *BPMN* Diagrammen kein Gegenstück gebraucht, stattdessen werden ***Sequence Flow***-Abbilder von ***Control Flows*** direkt an eine ***Activity***-Abbildung einer ***Function*** geführt. In *UML* Diagrammen wird ein ***Inclusive Merge Connector*** als ***Join Node*** mit inklusivem ***Join Specification***-Ausdruck (logisches *OR*) dargestellt (siehe Unterkapitel 3.3.2.3).

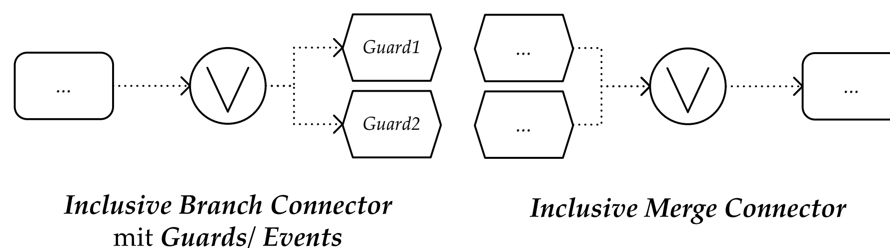


Abbildung 5.9.: Graphische Darstellung eines ***Inclusive Branch Connectors*** und eines ***Inclusive Merge Connectors***

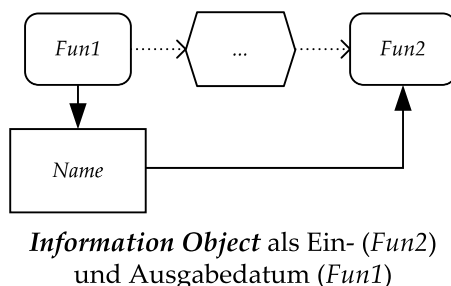
5.3.3. *Resources – Information Objects*

Information Objects definieren die Ein- bzw. Ausgabedaten von ***Functions*** (siehe Unterkapitel 5.3.1) und werden über gerichtete Kanten mit ***Functions*** verbunden. Der Unterschied zwischen Ein- und Ausgabedaten wird durch die Richtung des Pfeiles an der Kante deutlich: bei Eingabedaten steht der Pfeil auf der Seite der ***Function***, bei Ausgabedaten steht er an der Seite des ***Information Objects***.

Information Objects weisen Ähnlichkeiten zu *UML* ***Objects*** (siehe Unterkapitel 3.3.3) und *BPMN* ***Data Objects*** (siehe Unterkapitel 4.3.3) auf. Statt aber einem Daten-transportierenden ***Control Flow*** (siehe Unterkapitel 5.3.2.1) zugewiesen zu werden, wird ein ***Information Object*** über gerichtete Kanten (*Information Flow*) mit den ***Functions*** verbunden, welche einen Datensatz als Eingabe haben sowie denen, die diesen Datensatz als Ausgabe haben (siehe Abb. 5.10).

Eine vereinheitliche Repräsentation einer Informationseinheit existiert nicht, wodurch sich in der Literatur verschiede Darstellungen finden. In dieser Arbeit wird ein ***Information Object*** als ein

Rechteck dargestellt (siehe Abb. 5.10). Ein ***Information Object*** besitzt einen Bezeichner, im Gegensatz zum Bezeichner eines UML ***Objects*** aber in Anlehnung an den Bezeichner eines *BPMN* ***Data Objects*** setzt sich dieser nicht aus einem Instanznamen und einem Datentypen zusammen, sondern besteht lediglich aus einem allgemeiner gefassten Namen. Der Bezeichner wird in das Rechteck geschrieben (siehe Abb. 5.10). Die Angabe eines Zustandes ist nicht vorgesehen.



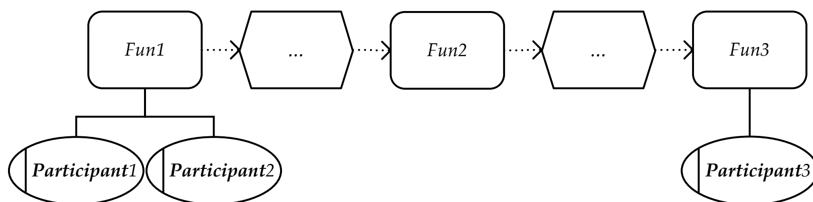
Information Object als Ein- (*Fun2*)
und Ausgabedatum (*Fun1*)

Abbildung 5.10.: Graphische Darstellung eines ***Information Objects***

5.3.4. *Participant Partitions – Organizational Units*

Eine ***Organizational Unit*** stellt einen Akteur in einem Geschäftsprozess (***Participant***) dar und entspricht den *UML* ***Activity Partitions*** (siehe Unterkapitel 3.3.4) und *BPMN* ***Swimlanes*** (siehe Unterkapitel 4.3.4). Über Zuordnungskanten lassen sich aktive, ausführbahre Elemente eines Prozesses (***Functions***, siehe Unterkapitel 5.3.1) einer oder mehreren ***Organizational Units*** zuweisen, von denen sie während der Laufzeit ausgeführt werden.

Eine in der Literatur (z.B. in [Sta06]) häufig anzutreffende graphische Repräsentation einer ***Organizational Unit***, besteht aus einem Oval, mit durch einen Strich abgetrennten, linken Bereich und dem Namen des ***Participants*** im rechten Bereich (siehe Abb. 5.11).



Die ***Functions*** *Fun1* und *Fun2* werden von den
Organizational Units (***Participant1*** und ***Participant2***) ausgeführt.
Die ***Function*** *Fun3* wird nur von ***Participant3*** ausgeführt.

Abbildung 5.11.: Graphische Darstellung von ***Organizational Units***

Da die Zuteilung von ***Functions*** zu immergleichen ***Organizational Units*** der Übersicht einer EPK schadet, kann auf eine wiederholte Zuordnung aufeinanderfolgender ***Functions*** zu einem gleichen ***Participant*** verzichtet werden (siehe Abb. 5.11). Sollten sich so entstehende Zuordnungsbereiche aufgrund von Verknüpfungsoperatoren überschneiden, werden die ***Functions*** in der Schnittmenge von den zuständigen ***Participants*** gemeinsam ausgeführt.

5.3.5. *Annotations – Descriptions*

Die Möglichkeit weitere Informationen/ Beschreibungen (***Descriptions***) über Notationselemente, den Ausführungsfluss, oder Implementierungsdetails als formale oder informale Annotation anzugeben ist von der EPK-Notation nicht vorgesehen. Modellierungswerkzeuge wie beispielsweise *ARIS Express*, der Firma *IDS Scheer*, ermöglichen die Speicherung von ***Descriptions*** als nicht-graphisch dargestellte Attribute, erlauben aber auch zusätzlich die Angabe von beliebigem Text an jedem beliebigen Notationselement. Über diesen Umweg lassen sich ***Descriptions*** trotzdem graphisch darstellen.

5.4. Fazit

Die EPK bietet eine umfangreiche Notation für Geschäftsprozesse im Allgemeinen und Geschäftsprozesse in *SOAMIG*, welche sich aber teils erheblich von den Sprachkonzepten der *UML* Aktivitätsdiagramme und denen der *BPMN* unterscheiden. Die Abbildung einer EPK auf ein *UML* Aktivitätsdiagramm oder ein *BPMN* Diagramm ist, wie auch die umgekehrte Abbildung, möglich aber an hohe Einschränkungen an den Modellierer und Anpassungen der EPK-Notation gebunden (siehe Kapitel 7).

Teil III.

Erstellung einer Referenz für UML Aktivitätsdiagramme, BPMN und EPK

6. Einleitung

In den vorherigen Kapiteln wurden die Sprachkonzepte und Notationselemente der drei Geschäftsprozessmodellierungssprachen, ausgerichtet auf die Bedürfnisse der Modellierung von Geschäftsprozessen im *SOAMIG*-Projekt, vorgestellt. Sprachelemente mit ähnlichen Zielsetzungen wurden dargestellt und kurz miteinander verglichen, auf Diskrepanzen wurde aber nicht weiter eingegangen (siehe Teil II).

Auf Grund der Unterschiede zwischen den Notationskonzepten muss für jede Modellierungsprache ein Konverter für die semantisch-äquivalente Transformation ersteller Geschäftsprozessmodelle in das von *JGraLab* verwertbare *TGraph*-Format erstellt werden (vgl. Unterkapitel 1.1). Um dem entgegenzuwirken wird in den folgenden Kapiteln auf die Unterschiede der Sprachkonzepte genauer eingegangen und Lösungen vorgestellt. Zudem wird für jede Elementkategorie und jeden Elementtyp ein Referenzkonzept vorgestellt, das die Eigenschaften der drei Sprachen in sich abbildet (siehe Unterkapitel 7).

Aus den gewonnenen Informationen werden anschließend passende Metamodelle für die drei Notationen und die Referenzsprache entwickelt und vorgestellt (siehe Unterkapitel 8). Mit Hilfe dieser Metamodelle lassen sich, in einer beliebigen Sprache und mit einem beliebigen Modellierungswerkzeug erhobene, zum jeweiligen Metamodell konforme, Geschäftsprozessmodelle anhand semantik-erhaltender Konvertierer in Metamodell-konforme Modelle der jeweils anderen Sprachen und in ein Referenzmetamodell-konformes Modell überführen (siehe Teil IV).

7. Vergleich der Sprachkonzepte und Notationselemente

Im folgenden Kapitel werden die in den vorherigen Kapiteln vorgestellten Sprachkonzepte in direkte Relation zueinander gebracht und Elemente mit gleicher Funktionalität durch Referenzelemente zusammengefasst. Es werden Gemeinsamkeiten und Diskrepanzen zwischen den Sprachkonzepten der verschiedenen Notationen sowie Lösungen beschrieben. Die Einteilung der Notationkonzepte basuert auf der eingeführten Orientierung an den Grundkonzepten aus Geschäftsprozessen (siehe Unterkapitel 2) sowie der damit einhergehenden Struktur der letzten Kapitel (siehe Unterkapitel 3.2, 4.2 und 5.2). Es wird unterschieden zwischen:

- ***Activities*** (siehe Unterkapitel 7.1),
- ***Flow Controls*** (siehe Unterkapitel 7.2),
- ***Resources*** (siehe Unterkapitel 7.3),
- ***Participant Partitions*** (siehe Unterkapitel 7.4) und
- ***Annotations*** (siehe Unterkapitel 7.5).

7.1. *Activities*

Vergleichbar zu:

- *UML* ***Activity Nodes*** (siehe Unterkapitel 3.3.1),
- *BPMN* ***Activities*** (siehe Unterkapitel 4.3.1) und
- EPK ***Functions*** (siehe Unterkapitel 5.3.1).

Beschreibung:

Activities beschreiben das Verhalten eines Geschäftsprozesses, werden über gerichtete Kanten in den Ausführungsfluss eingebunden (siehe Unterkapitel 7.2.2) und können Datensätze erhalten, bearbeiten sowie produzieren (siehe Unterkapitel 7.3). Bei ***Activities*** wird unterschieden zwischen:

- ***Atomic Process*** (siehe Unterkapitel 7.1.1),
- ***Sub-Process*** (siehe Unterkapitel 7.1.2) und
- ***Call Process*** (siehe Unterkapitel 7.1.3).

Unterschiede:
UML ***Activity Nodes***, *BPMN* ***Activities*** und EPK ***Functions*** beschreiben die gleiche Funktionalität. Unterschiede ergeben sich durch die Art wie ***Activities*** in der jeweiligen Notation in den Ausführungsfluss eingebunden werden, wann sie ausgeführt werden, sowie bei der Weise wie ihnen Datensätze zu- bzw. abeführt werden (siehe Unterkapitel 7.2.2 und Unterkapitel 7.3).

7.1.1. *Atomic Process*

Vergleichbar zu:

- *UML* ***Action*** (siehe Unterkapitel 3.3.1.1),
- *BPMN* ***Task*** (siehe Unterkapitel 4.3.1.1) und
- EPK ***Atomic Function*** (siehe Unterkapitel 5.3.1.1).

Beschreibung:
Atomic Processes modellieren einfache, kurze Funktionalitäten, sowie umfangreiche, nicht atomare Tätigkeiten, die jedoch im Modell/ Geschäftsprozess als atomar angesehen und nicht weiter aufgeteilt werden.

Unterschiede:
UML ***Actions***, *BPMN* ***Tasks*** und EPK ***Atomic Functions*** erfüllen die gleichen primären Aufgaben. ***Atomic Processes*** weisen, neben den von ***Activity*** geerbten Unterschieden zwischen den jeweiligen Ablegern der Notationen (siehe Unterkapitel 7.1), keine weiteren Diskrepanzen auf .

7.1.2. Sub-Process

Vergleichbar zu:

- *UML* ***Activity*** (siehe Unterkapitel 3.3.1.2),
- *BPMN* ***Sub-Process*** (siehe Unterkapitel 4.3.1.2) und
- EPK ***Complex Function*** (siehe Unterkapitel 5.3.1.2).

Beschreibung:
Sub-Processes modellieren das komplexe Verhalten nicht-atomarer Zusammenhänge oder ganzer Geschäftsprozesse und setzen sich aus weiteren Notationselementen zusammen.

Unterschiede:
UML ***Activities***, *BPMN* ***Sub-Processes*** und EPK ***Complex Function*** erfüllen die gleichen primären Aufgaben. ***Sub-Processes*** weisen, neben den von ***Activity*** geerbten Unterschieden zwischen den jeweiligen Ablegern der Notationen (siehe Unterkapitel 7.1), keine weiteren Diskrepanzen auf .

7.1.3. Call Process

Vergleichbar zu:

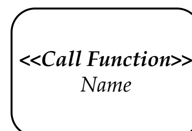
- *UML* ***Call Action*** (siehe Unterkapitel 3.3.1.3),
- *BPMN* ***Call Task*** (siehe Unterkapitel 4.3.1.3) und
- EPK ***Call Function*** (siehe Unterkapitel 5.3.1.3).

Beschreibung:
Call Processes stellen Platzhalter für ***Sub-Processes*** (siehe Unterkapitel 7.1.2) dar. Bei ihrer Ausführung wird das Verhalten des repräsentierten ***Sub-Process*** aufgerufen und die Ausführung an diesen übergeben.

Unterschiede:
UML ***Call Actions***, *BPMN* ***Call Tasks*** und EPK ***Call Functions*** erfüllen die gleichen primären Aufgaben. ***Call Processes*** weisen, neben den von ***Activity*** geerbten Unterschieden zwischen den jeweiligen Ablegern der Notationen (siehe Unterkapitel 7.1), kleinere Diskrepanzen auf:

- **Problemstellung:** Während sich *UML* ***Call Actions*** und *BPMN* ***Call Tasks*** graphisch von *UML* ***Activities*** bzw. *BPMN* ***Sub-Processes*** absetzen, besitzen EPK ***Call Functions*** kein graphisches Merkmal das sie von EPK ***Complex Functions*** unterscheidet.

  Lösung: Um eine sichtbare Differenzierung zwischen EPK ***Call Functions*** und EPK ***Complex Functions*** zu erhalten, wird dem Bezeichner einer ***Call Function*** der Stereotyp ***<<Call Function>>*** vorangestellt (siehe Abb. 7.1).



EPK ***Call Function***

Abbildung 7.1.: Der Stereotyp ***<<Call Function>>*** ermöglicht die bessere graphische Unterscheidung zwischen EPK ***Call Functions*** und EPK ***Complex Functions***.

7.2. Flow Controls

Vergleichbar zu:

- *UML* ***Flow Controls*** (siehe Unterkapitel 3.3.2),
- *BPMN* ***Flow Controls*** (siehe Unterkapitel 4.3.2) und
- EPK ***Flow Controls*** (siehe Unterkapitel 5.3.2).

Beschreibung:

Bei ***Flow Controls*** wird unterschieden zwischen:

- ***Events*** (siehe Unterkapitel 7.2.1),
- ***Flows*** (siehe Unterkapitel 7.2.2) und
- ***Connectors*** (siehe Unterkapitel 7.2.3).

Unterschiede:

UML ***Flow Controls***, *BPMN* ***Flow Controls*** und EPK ***Flow Controls*** beschreiben die gleiche Funktionalität. Unterschiede treten zwischen den sprachlichen Ablegern auf der Untertypenebene auf.

7.2.1. Events

Vergleichbar zu:

- *UML* ***Events*** (siehe Unterkapitel 3.3.2.2),
- *BPMN* ***Events*** (siehe Unterkapitel 4.3.2.2) und
- EPK ***Events*** (siehe Unterkapitel 5.3.2.2).

Beschreibung:

Events stellen das Auftreten eines, von außerhalb oder innerhalb des Diagrammes/ Geschäftsprozesses eingeläuteten, Herganges dar, der sich auf den Ausführungfluss auswirkt. Bei ***Events*** wird unterschieden zwischen:

- ***Start Events*** (siehe Unterkapitel 7.2.1.1),
- ***Flow Final Events*** (siehe Unterkapitel 7.2.1.2),
- ***Process Final Events*** (siehe Unterkapitel 7.2.1.3),
- ***Signal Throwing Events*** (siehe Unterkapitel 7.2.1.4) und
- ***Signal Catching Events*** (siehe Unterkapitel 7.2.1.5).

Unterschiede:

Während sich die Aufgabenstellung an *UML* ***Events*** und *BPMN* ***Events*** stark ähneln, weichen EPK ***Events*** hiervon leicht ab. EPK ***Events*** beschreiben nicht nur die Vorbedingung für die Ausführung einer EPK ***Function*** und den Ausgang einer EPK ***Function***, sondern erfüllen auch die Rolle von Ausführungsbedingungen im Kontrollfluss (siehe Unterkapitel 5.3.2.2 und 5.3.2.3). Eine klare Trennung zwischen EPK ***Events*** welche als Ausführungsbedingung dienen und solchen, die eine Signalsendung oder einen Signalsempfang darstellen, ist in der EPK-Notation nicht vorgesehen, während die *UML* Aktivitätsdiagramme und die *BPMN* hierfür mehrere ***Event***-Untertypen vorsehen. Während diese Probleme als typ-spezifisch angesehen werden können und daher in den folgenden Unterkapiteln behandelt werden, stellt sich ein allgemeineres Problem an alle ***Events***:

1. **Problemstellung**: Die EPK schreibt den verpflichtenden Einsatz eines Bezeichners an EPK ***Events*** vor, während die *UML* und *BPMN* die Vergabe eines Bezeichners für ***Events*** als optional ansehen.

   Lösung: In *UML* Aktivitätsdiagrammen und *BPMN* Diagrammen, die im Rahmen des *SOAMIG*-Projekts erstellt werden ist die Vergabe eines Bezeichners für *UML* ***Events*** und *BPMN* ***Events*** verpflichtend.
   Durch diese Maßnahme ergeben sich neue Probleme bei ***Start Events***, ***Process Final Events*** und ***Flow Final Events*** auf (siehe Unterkapitel 7.2.1.1, 7.2.1.3und 7.2.1.2).

2. **Problemstellung**: Die aus Kompatibilitätsgründen zur EPK verpflichtende Vergabe von Bezeichnern für *UML* ***Events*** und *BPMN* ***Events*** (siehe Problemstellung 1) stellt bei der Transformation bestimmter *UML* Aktivitätsdiagramme oder *BPMN* Diagramme in EPK-Notation ein Problem dar:

   a) Das Problem tritt auf, wenn der Ausführungsfluss über einen *UML* ***Initial Node***/ *BPMN* ***Start Event***, einen *UML* ***Accept Signal Event***/ BPMN ***Signal Catching Event*** oder einen Datensatz (siehe Unterkapitel 7.3) gestartet wird und dieses Element im Ausführungsfluss direkt von einer *UML* ***Activity***/ einem *BPMN* ***Sub-Process*** *X* gefolgt wird (siehe Abb. 7.2).

      Wird eine EPK als EPK ***Complex Function*** *X* – das Abbild der *UML* ***Activity***/ des *BPMN* ***Sub-Process*** *X* – in eine weitere EPK *Y* – das Abbild des *UML* Aktivitätsdiagrammes/ des *BPMN* Diagrammes – eingebunden, muss der ***Complex Function*** *X* an der eingesetzten Stelle in *Y* das/ die ***Start Events*** aus *X* vorangestellt werden (siehe Unterkapitel 5.3.1.2). Bei der Transformation eines Geschäftsprozesses mit oben beschriebenen Aufbau in die EPK-Notation würde direkt auf das ***Start Event*** *A* das ***Intermediate Event*** *B* – als Kopie des ***Start Events*** *B* aus der ***Complex Function*** *X* – folgen (siehe Abb. 7.2). Die EPK-Notation verbietet jedoch zwei aufeinander folgende ***Events***.

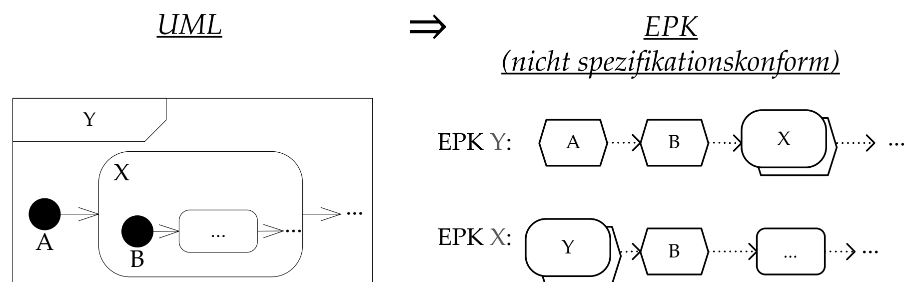


Abbildung 7.2.: Überführung eines *UML* Aktivitätsdiagrammes, nach dem in Problemstellung 2a beschriebenen Schema, in eine nicht-spezifikationskonforme EPK.

   Lösung:

   i. Bei der Transformation wird dem ***Intermediate Event*** B, an der Stelle an der die ***Complex Function*** *X* in der EPK *Y* eingebunden ist, eine automatisch generierte EPK ***Atomic Function*** F vorangeschrieben (siehe Abb. 7.3). Somit wechseln sich im

Ausführungsfluss, konform zur EPK-Spezifikation, immer EPK ***Events*** und EPK ***Functions*** ab.

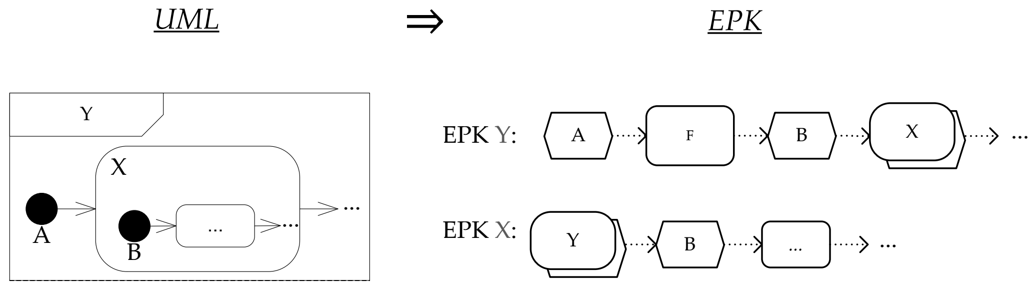


Abbildung 7.3.: Dem EPK ***Intermediate Event*** *B* wird, an der Stelle in der die EPK ***Complex Function*** *X* in der EPK *Y* eingebunden ist, eine bei der Transformation automatisch generierte EPK ***Atomic Function*** *F* vorangestellt.

ii. Alternativ zur Lösung i. kann das ***Start Event*** *A* auch in *Y* weggelassen werden und der Startpunkt durch die ***Complex Function*** *X* markiert werden. Diese Schreibweise wurde bei der Überführung des Einführungsbeispiels *sellTrainticket* angewandt (siehe Unterkapitel 5.2).

b) Das Problem tritt auf, wenn der Ausführungsfluss über einen *UML* ***Flow Final Node***/ *BPMN* ***End Event***, einen *UML* ***Activity Final Node***/ BPMN ***Signal Throwing Event*** oder einen Datensatz (siehe Unterkapitel 7.3) beendet wird und dieses Element im Ausführungsfluss direkt hinter einer *UML* ***Activity***/ einem *BPMN* ***Sub-Process*** *X* steht (siehe Abb. 7.4).

Wird eine EPK als EPK ***Complex Function*** *X* – das Abbild der *UML* ***Activity***/ des *BPMN* ***Sub-Process*** *X* – in eine weitere EPK *Y* – das Abbild des *UML* Aktivitätsdiagrammes/ des *BPMN* Diagrammes – eingebunden, muss der ***Complex Function*** *X* an der eingesetzten Stelle in *Y* das/ die ***End Events*** aus *X* nachgestellt werden (siehe Unterkapitel 5.3.1.2). Bei der Transformation eines Geschäftsprozesses mit oben beschriebenem Aufbau in die EPK-Notation würde direkt auf das ***Intermediate Event*** *B* – als Kopie des ***End Events*** *B* aus der ***Complex Function*** *X* – das ***End Event*** *A* folgen (siehe Abb. 7.4). Die EPK-Notation verbietet jedoch zwei aufeinander folgende ***Events***.

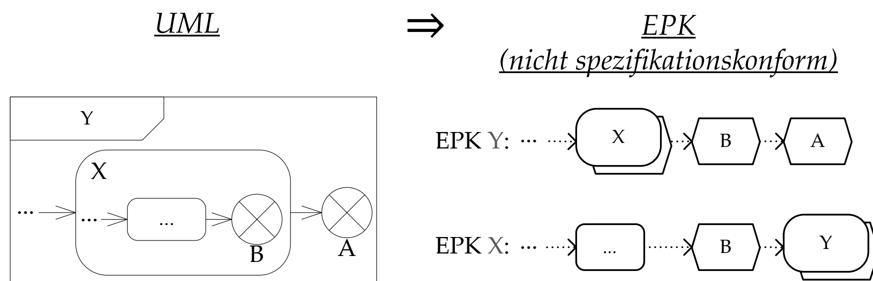


Abbildung 7.4.: Überführung eines *UML* Aktivitätsdiagrammes, nach dem in Problemstellung 2b beschriebenen Schema, in eine nicht-spezifikationskonforme EPK.

Lösung:

Bei der Transformation wird dem ***Intermediate Event*** *A*, an der Stelle an der die ***Complex Function*** *X* in der EPK *Y* eingebunden ist, eine automatisch generierte EPK ***Atomic Function*** F nachgestellt (siehe Abb. 7.5). Somit wechseln sich im Ausführungsfluss, konform zur EPK-Spezifikation, immer EPK ***Events*** und EPK ***Functions*** ab.

UML ⇒ *EPK*

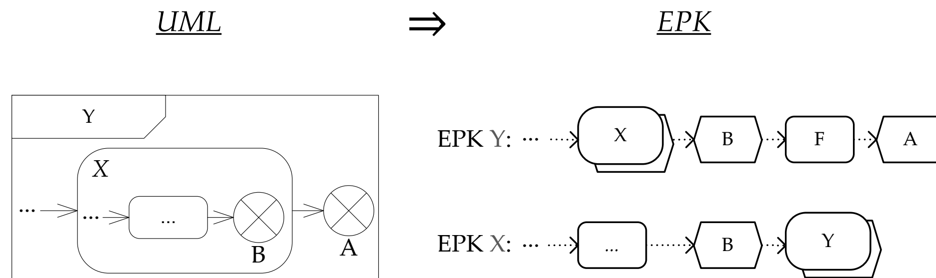


Abbildung 7.5.: Dem EPK ***Intermediate Event*** *B* wird, an der Stelle in der die EPK ***Complex Function*** *X* in der EPK *Y* eingebunden ist, eine bei der Transformation automatisch generierte EPK ***Atomic Function*** *F* nachgestellt.

7.2.1.1. Start Event

Vergleichbar zu:

- *UML* ***Initial Node*** (siehe Unterkapitel 3.3.2.2),
- *BPMN* ***Start Event*** (siehe Unterkapitel 4.3.2.2) und
- EPK ***Start Event*** (siehe Unterkapitel 5.3.2.2).

Beschreibung:

Start Events legen die Stellen in einem Diagramm fest, an denen die Ausführung beginnt. Besitzt ein Diagramm mehr als ein ***Start Event***, werden mehrere unabhängige, parallel verlaufende Ausführungsflüsse gestartet.

Unterschiede:

UML ***Initial Nodes***, *BPMN* ***Start Events*** und EPK ***Start Events*** verfolgen die gleiche Zielsetzung. Dennoch existieren einige Inkompatibilitäten zwischen den *UML*, *BPMN* und EPK Ablegern dieses Typs, welche im Unterkapitel 7.2.1 und ergänzend an dieser Stelle beschrieben werden:

1. **Problemstellung**: Während *UML* Aktivitätsdiagramme und *BPMN* Diagramme ohne *UML* ***Initial Nodes*** bzw. *BPMN* ***Start Events*** modelliert werden können, setzt die EPK-Notation den Einsatz mindestens eines EPK ***Start Events*** in einer EPK als verpflichtend voraus. In *UML* Aktivitätsdiagrammen und *BPMN* Diagrammen können die Stellen, an denen die Ausführung des Diagrammes beginnen soll, auch alternativ durch *UML* ***Accept Signal Actions***

bzw. *BPMN* ***Signal Catching Events*** oder *UML* ***Object Nodes*** bzw. *BPMN* ***Data Objects*** (siehe Unterkapitel 7.3) festgelegt werden.

Lösung: In Anlehnung an die, aus *UML* und *BPMN* beschriebene, alternative Möglichkeit die Startpunkte eines Diagrammes festzulegen, werden für EPK die ***Event***-Untertypen ***Signal Catching Event*** und ***Resource Accepting Event*** eingeführt (siehe Unterkapitel 7.2.1.5 und 7.3). Diese Spezialisierungen entsprechen weitestgehend gewöhnlichen ***Start Events*** und können an der ihnen vorgesehenen Stelle am Anfang der Ausführungsflüsse von EPK eingesetzt werden.

2. **Probemstellung**: Die *UML*-Spezifikation schreibt für *UML* ***Initial Node*** vor, dass alle ausgehenden *UML* ***Activity Edges*** vom Untertyp *UML* ***Control Flow*** sein müssen (vgl.**[OMG09b]**). Die *BPMN*- und EPK-Spezifikationen schreiben diesbezüglich keine Einschränkungen vor.

   Lösung: In Anlehnung an die *UML*-Notation dürfen die aus *BPMN* ***Start Events*** bzw. EPK ***Start Events*** ausgehenden *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** keine Daten transportieren. Gleiches gilt in Referenzmodellen, in denen alle aus ***Start Events*** ausgehende Kanten vom Typ ***Flow*** sein müssen (siehe Unterkapitel 7.2.2.1).

7.2.1.2. Flow Final Event

Vergleichbar zu:

- *UML* ***Flow Final Node*** (siehe Unterkapitel 3.3.2.2),
- *BPMN* ***End Event*** (siehe Unterkapitel 4.3.2.2) und
- EPK ***End Event*** (siehe Unterkapitel 5.3.2.2).

Beschreibung:

Flow Final Events und ***Process Final Events*** bilden die Gegenstücke zu ***Start Events***. Beide modellieren die geplante Beendigung eines Ausführungsflusses und können zusammen und in mehreren Ausführungen in einem Diagramm vorkommen. Ein ***Flow Final Event*** beendet nur den eingehenden Fluss, die restlichen Ausführungsflüsse laufen unberührt weiter.

Unterschiede:

UML ***Flow Final Nodes***, *BPMN* ***End Events*** und EPK ***End Events*** verfolgen die gleiche Zielsetzung. Dennoch existieren einige Inkompatibilitäten zwischen den *UML*, *BPMN* und EPK Ablegern dieses Typs, welche im Unterkapitel 7.2.1 und ergänzend an dieser Stelle beschrieben werden:

1. **Problemstellung**: Während *UML* Aktivitätsdiagramme und *BPMN* Diagramme ohne Endereignisse modelliert werden können, setzt die EPK-Notation den Einsatz mindestens eines EPK ***End Events*** oder EPK ***Terminate End Events*** (siehe Unterkapitel 7.2.1.3) in einer EPK als verpflichtend voraus. In *UML* Aktivitätsdiagrammen und *BPMN* Diagrammen können die Stellen, an denen die Ausführung eines Diagrammes enden soll, auch alternativ durch *UML* ***Send Signal Actions*** bzw. *BPMN* ***Signal Throwing Events*** oder *UML* ***Object Nodes*** bzw. *BPMN* ***Data Objects*** (siehe Unterkapitel 7.3) festgelegt werden.

Lösung: In Anlehnung an die, aus *UML* und *BPMN* beschriebene, alternative Möglichkeit die Endpunkte eines Diagrammes festzulegen, werden für EPK die ***Event***-Untertypen ***Signal Throwing Event*** und ***Resource Passing Event*** eingeführt (siehe Unterkapitel 7.2.1.4 und 7.3). Diese Spezialisierungen entsprechen weitestgehend gewöhnlichen ***End Events*** und können an der ihnen vorgesehenen Stelle am Ende der Ausführungsflüsse von EPK eingesetzt werden.

2. **Problemstellung**: Die *UML*-Spezifikation schreibt für *UML* ***Flow Final Nodes*** vor, dass alle eingehenden *UML* ***Activity Edges*** vom Untertyp *UML* ***Control Flow*** sein müssen (vgl. **[OMG09b]**). Die *BPMN*- und EPK-Spezifikationen schreiben diesbezüglich keine Einschränkungen vor.

   Lösung: In Anlehnung an die *UML*-Notation dürfen die in *BPMN* ***End Events*** bzw. EPK ***End Events*** eingehenden *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** keine Daten transportieren. Gleiches gilt in Referenzmodellen, in denen alle aus ***End Events*** ausgehende Kanten vom Typ ***Flow*** sein müssen (siehe Unterkapitel 7.2.2.1).

7.2.1.3. Process Final Event

Vergleichbar zu:

- *UML* ***Activity Final Node*** (siehe Unterkapitel 3.3.2.2) und
- *BPMN* ***Terminate End Event*** (siehe Unterkapitel 4.3.2.2) während
- EPK kein direktes Gegenstück besitzen (siehe Paragraph Unterschiede, Problemstellung 1).

Beschreibung:

Process Final Events und ***Flow Final Events*** bilden die Gegenstücke zu ***Start Events***. Beide modellieren die geplante Beendigung eines Ausführungsflusses und können zusammen und in mehreren Ausführungen in einem Diagramm vorkommen. Ein ***Process Final Node*** beendet alle Ausführungsflüsse und damit die Ausführung des gesamten Diagramms/ Geschäftsprozesses.

Unterschiede:

Während die *UML* mit den *UML* ***Activity Final Nodes*** und die *BPMN* mit den *BPMN* ***Terminate End Events*** die gleiche Funktionalität anbieten, ergeben sich bei der EPK-Notation zu diesem Kontext einige Komplikationen:

1. **Problemstellung**: Durch zwei unterschiedliche Syntaxelemente ermöglichen die *UML*- und *BPMN*-Spezifikationen die explizite Unterscheidung zwischen Endereignissen, welche nur den eingehenden Ausführungsfluss und Endereignissen, welche die Ausführung des gesamten Diagrammes beenden. Die EPK bietet hingegen kein Syntaxelement um die vollständige Ausführung eines Diagrammes explizit zu beenden.

   Lösung: Die EPK-Notation wird um das Notationselement ***Terminate End Event*** erweitert. ***Terminate End Events*** modellieren in, im *SOAMIG*-Rahmen erstellten, Geschäftsprozessmodellen die Terminierung des gesamten Geschäftsprozesses. Graphisch unterscheiden sich

Terminate End Events dahingegen von gewöhnlichen EPK ***End Events***, dass ihrem Bezeichner der Stereotyp **<<*Terminate End Event*>>** vorangestellt wird (siehe Abb. 7.6).

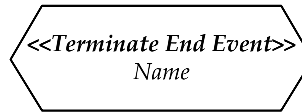


Abbildung 7.6.: Erweiterung der EPK-Notation um die explizit von gewöhnlichen EPK ***End Events*** unterscheidbaren EPK ***Terminate End Events***, welche bei einlaufendem Ausführungsfluss die Ausführung des gesamten Modelles beenden.

2. **Problemstellung**: Während *UML* Aktivitätsdiagramme und *BPMN* Diagramme ohne Endereignisse modelliert werden können, setzt die EPK-Notation den Einsatz mindestens eines EPK ***Terminate End Events*** oder EPK ***End Events*** in einer EPK als verpflichtend voraus. In *UML* Aktivitätsdiagrammen und *BPMN* Diagrammen können die Stellen, an denen die Ausführung eines Diagrammes enden soll, auch alternativ durch *UML* ***Send Signal Actions*** bzw. *BPMN* ***Signal Throwing Events*** oder *UML* ***Object Nodes*** bzw. *BPMN* ***Data Objects*** (siehe Unterkapitel 7.3) festgelegt werden.

   Lösung: Diese Problemstellung und Lösung sind identisch mit der Problemstellung und Lösung aus dem Unterkapitel 7.2.1.2.

3. **Problemstellung**: Die *UML*-Spezifikation schreibt für *UML* ***Activity Final Node*** vor, dass alle eingehenden *UML* ***Activity Edges*** vom Untertyp *UML* ***Control Flow*** sein müssen (vgl. **[OMG09b]**). Die *BPMN*- und EPK-Spezifikationen schreiben diesbezüglich keine Einschränkungen vor.

   Lösung: In Anlehnung an die *UML*-Notation dürfen die in *BPMN* ***Terminate End Events*** bzw. EPK ***Terminate End Events*** eingehenden *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** keine Daten transportieren. Gleiches gilt in Referenzmodellen, in denen alle aus ***Terminate End Events*** ausgehende Kanten vom Typ ***Flow*** sein müssen (siehe Unterkapitel 7.2.2.1).

7.2.1.4. Signal Throwing Event

Vergleichbar zu:

- *UML* ***Send Signal Action*** (siehe Unterkapitel 3.3.2.2) und
- *BPMN* ***Signal Throwing Event*** (siehe Unterkapitel 4.3.2.2), während
- EPK kein direktes Gegenstück besitzen (siehe Paragraph Unterschiede, Problemstellung 1).

Beschreibung:

Signal Throwing Events lösen das Senden eines asynchronen Signals, an ein oder mehrere ***Signal Catching Events*** bei eintreffendem Ausführungsfluss, aus. Nach Auslösen des Signals läuft der Ausführungsfluss am ***Signal Throwing Event*** weiter, parallel werden weitere, unabhängige Ausführungsflüsse an den ***Signal Catching Events*** gestartet. Ein ***Signal Throwing Event*** kann ein Signal auch mehrmals verschicken.

Unterschiede:

Während die *UML* und die *BPMN* mit den *UML* ***Send Signal Actions*** bzw. den *BPMN* ***Signal Throwing Events*** die gleiche Funktionalität anbieten, bereitet die EPK-Notation ein Problem bei diesem Konzept:

1. **Problemstellung**: Die EPK-Notation sieht keine explizite Differenzierung zwischen gewöhnlichen ***Intermediate Events***/ ***End Events*** und solchen die ein Signal versenden vor.

   Lösung: Die EPK-Notation wird um die Notationselemente ***Signal Throwing Intermediate Event*** und ***Signal Throwing End Event*** erweitert, welche zusammen als ***Signal Throwing Events*** bezeichnet werden. Hierbei handelt es sich um Spezialisierungen von ***Intermediate Events*** und ***End Events***, welche explizit die gleiche Funktionalität wie *UML* ***Send Signal Actions*** und *BPMN* ***Signal Throwing Events*** aufweisen. Graphisch unterscheiden sich ***Signal Throwing Events*** dahingegen von gewöhnlichen ***Intermediate Events*** und ***End Events***, dass ihrem Bezeichner der Stereotyp ***<<Signal Throwing Event>>*** vorangestellt wird (siehe Abb. 7.7).

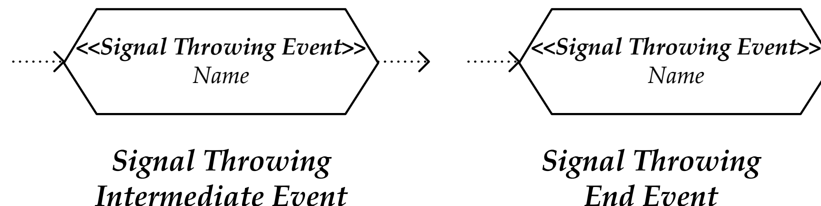


Abbildung 7.7.: Erweiterung der EPK-Notation um die explizit von gewöhnlichen EPK ***Intermediate Events*** und EPK ***End Events*** unterscheidbaren ***Signal Throwing Events***: ***Signal Throwing Intermediate Event*** und ***Signal Throwing End Event***.

7.2.1.5. Signal Catching Event

Vergleichbar zu:

- *UML* ***Accept Signal Action*** (siehe Unterkapitel 3.3.2.2) und
- *BPMN* ***Signal Catching Event*** (siehe Unterkapitel 4.3.2.2), während
- EPK kein direktes Gegenstück besitzen (siehe Paragraph Unterschiede, Problemstellung 1).

Beschreibung:

Ein ***Signal Catching Event*** wird durch den Empfang eines asynchronen Signals ausgelöst, das durch ein oder mehrere ***Signal Throwing Events*** von innerhalb oder außerhalb des aktuellen Diagrammes geschickt wurde. Ein ***Signal Catching Event***, das am Anfang eines Diagrammes steht, also keine eingehenden Ausführungsflüsse besitzt, ist ab Ausführungsbeginn bereit, Signale zu empfangen. Ein ***Signal Catching Event*** inmitten des Diagrammes wird hingegen erst für den Signalempfang aktiviert wenn der erste Ausführungsfluss eingeht. Bei Eingang eines Signals wird ein neuer Ausführungsfluss gestartet, welcher parallel und unabhängig zu anderen Ausführungsflüssen verläuft. Ein ***Signal Catching Event*** kann mehrmals ausgelöst werden.

Unterschiede:

Während die *UML* und die *BPMN* mit den *UML* ***Accept Signal Actions*** bzw. den *BPMN* ***Signal Catching Events*** die selbe Funktionalität zum Empfang von Signalen anbieten, bereitet die EPK-Notation bei diesem Konzept ein Problem:

1. **Problemstellung**: Die EPK-Notation sieht keine explizite Differenzierung zwischen gewöhnlichen ***Start Events***/ ***Intermediate Events*** und solchen die ein Signal empfangen vor.

   Lösung: Die EPK-Notation wird um die Notationselemente ***Signal Catching Start Event*** und ***Signal Catching Intermediate Event*** erweitert, welche zusammen als ***Signal Catching Events*** bezeichnet werden. Hierbei handelt es sich um Spezialisierungen von ***Start Events*** und ***Intermediate Events***, welche explizit die gleiche Funktionalität wie *UML* ***Accept Signal Actions*** und *BPMN* ***Signal Catching Events*** aufweisen. Graphisch unterscheiden sich ***Signal Catching Events*** dahingegen von gewöhnlichen ***Start Events*** und ***Intermediate Events***, dass ihrem Bezeichner der Stereotyp ***<<Signal Catching Event>>*** vorangestellt wird (siehe Abb. 7.8).

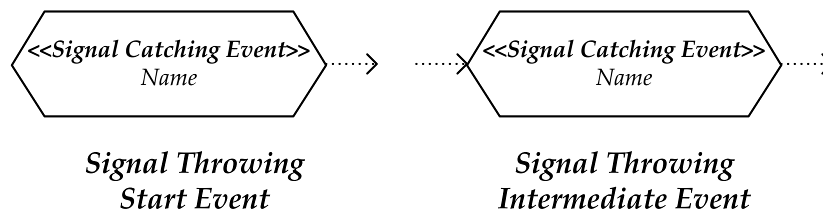


Abbildung 7.8.: Erweiterung der EPK-Notation um die explizit von gewöhnlichen EPK ***Start Events*** und EPK ***Intermediate Events*** unterscheidbaren ***Signal Catching Events***: ***Signal Catching Intermediate Event*** und ***Signal Catching End Event***.

7.2.2. Flow

Vergleichbar zu:

- *UML* ***Control Flow*** (siehe Unterkapitel 3.3.2.1),
- *BPMN* ***Sequence Flow*** (siehe Unterkapitel 4.3.2.1), und
- EPK ***Control Flow*** (siehe Unterkapitel 5.3.2.1).

Beschreibung:

Flows modellieren gerichtete Verbindungen zwischen ***Activities*** (siehe Unterkapitel 7.1), ***Events*** (siehe Unterkapitel 7.2.1) sowie ***Connectors*** (siehe Unterkapitel 7.2.3) und legen die Ablaufreihenfolge eines Diagrammes fest.

Unterschiede:

Obwohl die Grundidee hinter *UML* ***Control Flows***, *BPMN* ***Sequence Flows*** und EPK ***Control Flows*** die gleiche ist, unterscheiden sie sich in mehreren Hinsichten voneinander:

1. **Problemstellung**: Die Notationen unterscheiden sich beim Ausführungszeitpunkt ihrer Sprachelemente. Während die Ausführung eines *UML* ***Activity Nodes*** und *UML* ***Signal Events*** mit mehreren eingehenden *UML* ***Activity Edges*** erst startet, wenn der Ausführungsfluss an allen *UML* ***Activity Edges*** anliegt (*AND*-Verknüpfung), startet die Ausführung einer *BPMN* ***Activity*** und eines *BPMN* ***Events*** mit mehreren eingehenden *BPMN* ***Sequence Flows***, sobald der Ausführungsfluss an einem oder mehreren eingehenden *BPMN* ***Sequence Flows*** anliegt (*OR*-Verknüpfung). Die Ausführung eines *UML* ***Flow Final Nodes*** und *UML* ***Activity Final Nodes*** mit mehreren eingehenden *UML* ***Activity Edges*** startet hingegen sobald der Ausführungsfluss, ab einem oder mehreren eingehenden UML Activity Edges anliegt (*OR*-Verknüpfung). Die EPK erlaubt hingegen nur maximal einen eingehenden EPK ***Control Flow*** an einer EPK ***Function*** und einem EPK ***Event***.

   Lösung: Die Lösung ergibt sich aus mehreren Einzellösungen, jeweils für die Transformation eines Diagrammes aus einer Sprache in ein Diagramm der jeweils anderen Sprachen.

   a) Wird ein *UML* Aktivitätsdiagramm auf ein *BPMN* Diagramm abgebildet, werden mehrere eingehende *UML* ***Activity Edges*** eines *UML* ***Activity Nodes*** oder eines *UML* ***Signal Events*** in entsprechende *BPMN* ***Sequence Flow*** Abbilder transformiert und über ein *BPMN* ***Parallel Merge Gateway*** zusammengefasst. Ein zusätzlicher ***Sequence Flow*** verbindet den ***Parallel Merge Gateway*** mit dem entsprechenden *BPMN* ***Activity*** Abbild des *UML* ***Activity Nodes*** oder *BPMN* ***Signal Event*** Abbild des *UML* ***Signal Events*** (siehe Abb. 7.9).

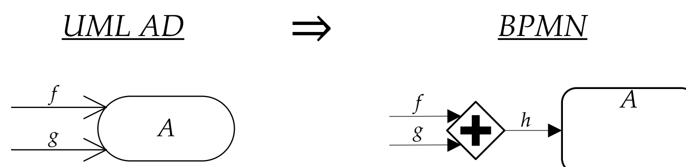


Abbildung 7.9.: Überführung eines *UML* ***Activity Nodes*** mit mehreren eingehenden *UML* ***Activity Edges*** in *BPMN*-Notation. Die Überführung eines *UML* ***Signal Events*** in *BPMN*-Notation hat die gleichen Züge.

   b) Wird ein *UML* Aktivitätsdiagramm auf ein *BPMN* Diagramm abgebildet, werden mehrere eingehende *UML* ***Activity Edges*** eines *UML* ***Flow Final Nodes*** oder eines *UML* ***Activity Final Nodes*** in entsprechende *BPMN* ***Sequence Flow*** Abbilder transformiert. Weil mehrere in ein *UML* ***Flow Final Node*** oder *UML* ***Activity Final Node*** eingehende *UML* ***Activity Edges*** eine *OR*-Verknüpfung darstellen, muss bei der Überführung in die BPMN-Notation kein *BPMN* ***Inclusive Merge Gateway*** erstellt werden. Die *BPMN* ***Sequence Flow*** Abbilder der *UML* ***Activity Edges*** fließen direkt in das *BPMN* ***End Event*** Abbild des *UML* ***Flow Final Nodes*** oder *BPMN* ***Terminate End Event*** Abbild des *UML* ***Activity Final Nodes*** (siehe Abb. 7.10).

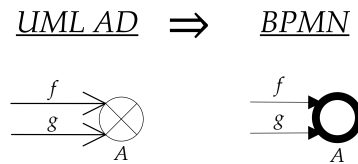


Abbildung 7.10.: Überführung eines *UML* ***Flow Final Nodes*** mit mehreren eingehenden *UML* ***Activity Edges*** in *BPMN*-Notation. Die Überführung eines *UML* ***Activity Final Nodes*** in *BPMN*-Notation hat die gleichen Züge.

c) Wird ein *UML* Aktivitätsdiagramm in eine EPK überführt, werden die eingehenden *UML* ***Activity Edges*** eines *UML* ***Activity Nodes*** oder eines *UML* ***Signal Events*** in entsprechende EPK ***Control Flow*** Abbilder transformiert und über einen EPK ***Parallel Merge Connector*** zusammengeführt. Ein zusätzlicher EPK ***Control Flow*** verbindet den EPK ***Parallel Merge Connector*** mit dem entsprechenden EPK ***Function*** Abbild des *UML* ***Activity Nodes*** oder dem EPK ***Signal Event*** Abbild des *UML* ***Signal Events*** (siehe Abb. 7.11).

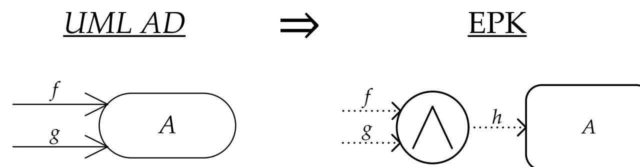


Abbildung 7.11.: Überführung eines *UML* ***Activity Nodes*** mit mehreren eingehenden *UML* ***Activity Edges*** in EPK-Notation. Die Überführung eines *UML* ***Signal Events*** in EPK-Notation hat die gleichen Züge.

d) Wird ein *UML* Aktivitätsdiagramm auf eine EPK abgebildet, werden mehrere eingehende *UML* ***Activity Edges*** eines *UML* ***Flow Final Nodes*** oder eines *UML* ***Activity Final Nodes*** in entsprechende *EPK* ***Control Flow*** Abbilder transformiert. Weil mehrere in ein *UML* ***Flow Final Node*** oder *UML* ***Activity Final Node*** eingehende *UML* ***Activity Edges*** eine *OR*-Verknüpfung darstellen, muss bei der Überführung in die EPK-Notation kein *EPK* ***Inclusive Merge Connector*** erstellt werden. Die EPK ***Control Flow*** Abbilder der *UML* ***Activity Edges*** fließen direkt in das EPK ***End Event*** Abbild des *UML* ***Flow Final Nodes*** oder EPK ***Terminate End Event*** Abbild des *UML* ***Activity Final Nodes*** (siehe Abb. 7.12).

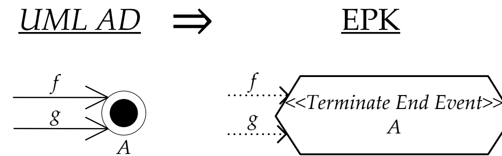


Abbildung 7.12.: Überführung eines *UML* ***Activity Final Nodes*** mit mehreren eingehenden *UML* ***Activity Edges*** in EPK-Notation. Die Überführung eines *UML* ***Flow Final Nodes*** in EPK-Notation hat die gleichen Züge.

e) Wird ein *BPMN* Diagramm auf ein *UML* Aktivitätsdiagramm abgebildet, werden mehrere eingehende *BPMN* ***Sequence Flows*** einer *BPMN* ***Activity*** oder eines *BPMN* ***Signal Events*** in entsprechende *UML* ***Activity Edge*** Abbilder transformiert und über ein *UML* ***Join Node*** mit inklusivem ***Join Specification***-Ausdruck zusammengeführt. Eine zusätzliche *UML* ***Activity Edge*** verbindet den *UML* ***Join Node*** mit dem entsprechenden *UML* ***Activity Node*** Abbild der *BPMN* ***Activity*** oder dem *UML* ***Signal Event*** Abbild des *BPMN* ***Signal Events*** (siehe Abb. 7.13).

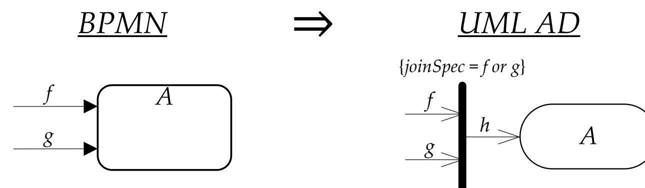


Abbildung 7.13.: Überführung einer *BPMN* ***Activity*** mit mehreren eingehenden *BPMN* ***Sequence Flows*** in *UML* Notation. Die Überführung eines *BPMN* ***Signal Events*** in EPK-Notation hat die gleichen Züge.

f) Wird ein *BPMN* Diagramm auf ein *UML* Aktivitätsdiagramm abgebildet, werden mehrere eingehende *BPMN* ***Sequence Flows*** eines *BPMN* ***End Events*** oder eines *BPMN* ***Terminate End Events*** in entsprechende *UML* ***Activity Edge*** Abbilder transformiert. Da mehrere in ein *UML* ***Flow Final Node*** oder *UML* ***Activity Final Node*** eingehende *UML* ***Activity Edges***, wie auch mehrere eingehende *BPMN* ***Sequence Flows*** in eine *BPMN* ***Activity*** oder ein *BPMN* ***Event*** eine *OR*-Verknüpfung darstellen, muss bei der Überführung des *BPMN* Diagrammes in *UML*-Notation kein ***Join Node*** mit inklusivem ***Join Specification***-Ausdruck erstellt werden, um die Flüsse zusammenzuführen. Die *UML* ***Activity Edge*** Abbilder der *BPMN* ***Sequence Flows*** können direkt in das *UML* ***Flow Final Node*** Abbild des *BPMN* ***End Events*** oder *UML* ***Activity Final Node*** Abbild des *BPMN* ***Terminate End Events*** einfließen (siehe Abb. 7.14).

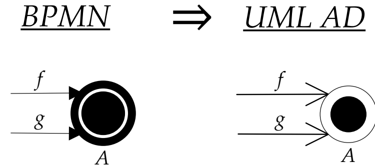


Abbildung 7.14.: Überführung einer *BPMN* ***Terminate End Events*** mit mehreren eingehenden *BPMN* ***Sequence Flows*** in *UML*-Notation. Die Überführung eines *BPMN* ***End Events*** in *UML*-Notation hat die gleichen Züge.

g) Wird ein *BPMN* Diagramm in eine EPK überführt, werden die eingehenden *BPMN* ***Sequence Flows*** einer *BPMN* ***Activity*** oder eines *BPMN* ***Events*** in entsprechende EPK ***Control Flow*** Abbilder transformiert und über einen EPK ***Inclusive Merge Connector*** zusammengeführt. Ein zusätzlicher EPK ***Control Flow*** verbindet den EPK ***Inclusive Merge Connector*** mit dem entsprechenden EPK ***Function*** Abbild der *BPMN* ***Activity*** oder dem EPK ***Event*** Abbild des *BPMN* ***Events*** (siehe Abb. 7.15).

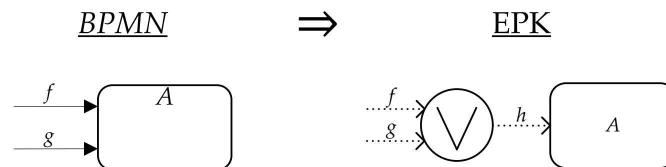


Abbildung 7.15.: Überführung einer *BPMN* ***Activity*** mit mehreren eingehenden *BPMN* ***Sequence Flows*** in EPK-Notation. Die Überführung eines *BPMN* ***Events*** in EPK-Notation hat die gleichen Züge.

h) Wird eine EPK auf ein *UML* Aktivitätsdiagramm abgebildet, in der ein EPK ***Parallel Merge Connector*** im Ausführungsfluss direkt vor einer EPK ***Function*** oder einem EPK ***Signal Event*** steht, dann wird dieser EPK ***Parallel Merge Connector*** bei der Überführung in die *UML*-Notation weggelassen. Die *UML* ***Control Flow*** Abbilder der EPK ***Control Flows*** die in diesen ***Parallel Merge Connector*** eingehen, laufen direkt in das *UML* ***Activity Node*** Abbild der EPK ***Function*** oder das *UML* ***Signal Event*** Abbild des EPK ***Signal Events***. Der EPK ***Control Flow*** zwischen ***Parallel Merge Connector*** und EPK ***Function*** oder EPK ***Signal Event*** wird nicht im *UML* Aktivitätsdiagramm abgebildet (siehe Abb. 7.16).

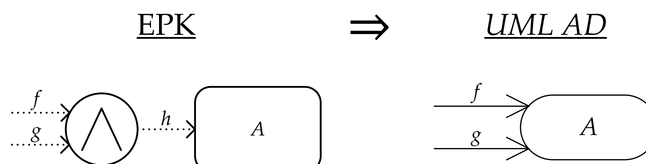


Abbildung 7.16.: Überführung einer EPK ***Function*** mit mehreren eingehenden EPK ***Control Flows*** in *UML*-Notation. Die Überführung eines EPK ***Signal Events*** in *UML*-Notation hat die gleichen Züge.

i) Wird eine EPK auf ein *UML* Aktivitätsdiagramm abgebildet, in dem ein EPK ***Inclusive Merge Connector*** im Ausführungsfluss direkt vor einem EPK ***End Event*** oder EPK ***Terminate End Event*** steht, dann muss bei der Transformation in die *UML*-Notation kein *UML* ***Join Node*** mit inklusivem ***Join Specification***-Ausdruck zur Zusammenführung der Ausführungsflüsse vor dem *UML* ***Flow Final Node*** Abbild des EPK ***End Events*** oder *UML* ***Activity Final Node*** Abbild des EPK ***Terminate End Events*** erstellt werden (siehe Abb. 7.17). Mehrere eingehende Flüsse in ein *UML* ***Flow Final Node*** oder *UML* ***Activity Final Node*** stellen eine implizite *OR*-Verknüpfung dar.

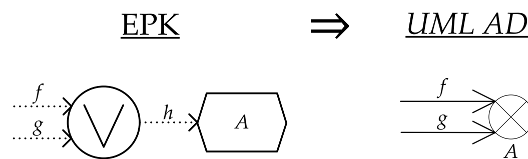


Abbildung 7.17.: Überführung eines EPK ***End Events*** mit mehreren eingehenden EPK ***Control Flows*** in *UML*-Notation. Die Überführung eines EPK ***Terminate End Events*** in *UML*-Notation hat die gleichen Züge.

j) Wird eine EPK auf ein *BPMN* Diagramm abgebildet, in der ein EPK ***Inclusive Merge Connector*** im Ausführungsfluss direkt vor einer EPK ***Function*** oder einem EPK ***Event*** steht, dann wird dieser EPK ***Inclusive Merge Connector*** bei der Überführung in die *BPMN*-Notation weggelassen. Die *BPMN* ***Sequence Flow*** Abbilder der EPK ***Control Flows*** die in diesen ***Inclusive Merge Connector*** eingehen, laufen direkt in das *BPMN* ***Activity*** Abbild der EPK ***Function*** oder das BPMN ***Event*** Abbild des EPK ***Events***. Der EPK ***Control Flow*** zwischen ***Inclusive Merge Connector*** und EPK ***Function*** oder EPK ***Event*** wird nicht im *BPMN* Diagramm abgebildet (siehe Abb. 7.16).

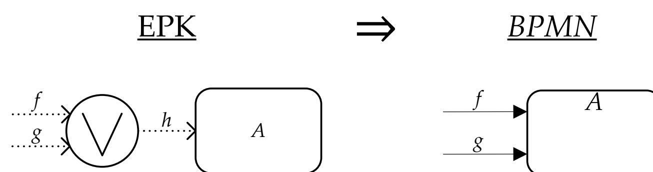


Abbildung 7.18.: Überführung einer EPK ***Function*** mit mehreren eingehenden EPK ***Control Flows*** in *BPMN*-Notation. Die Überführung eines EPK ***Events*** in *BPMN*-Notation hat die gleichen Züge.

2. **Problemstellung**: Nach Abarbeitung einer *UML* ***Activity Node*** oder eines *UML* ***Events*** mit mehreren ausgehenden *UML* ***Activity Edges***, wie auch nach Abarbeitung einer *BPMN* ***Activity*** oder eines *BPMN* ***Events*** mit mehreren ausgehenden *BPMN* ***Sequence Flows***, wird der Ausführungsfluss auf mehrere unabhängige, parallele Ausführungszweige aufgeteilt (*AND*-Aufteilung). Die EPK-Notation erlaubt hingegen nur maximal einen ausgehenden EPK ***Control Flow*** für EPK ***Functions*** und EPK ***Events***.

Lösung: Wird ein *UML* Aktivitätsdiagramm oder ein *BPMN* Diagramm in eine EPK überführt, wird den EPK ***Control Flow*** Abbildern der ausgehenden *UML* ***Activity Edges*** eines *UML* ***Activity Nodes*** oder *UML* ***Events*** bzw. den ausgehenden *BPMN* ***Sequence Flows*** einer *BPMN* ***Activity*** oder eines *BPMN* ***Events*** ein EPK ***Parallel Merge Connector*** vorangestellt. Ein zusätzlicher EPK ***Control Flow*** verbindet das entsprechende EPK ***Function*** Abbild des *UML* ***Activity Nodes*** bzw. der *BPMN* ***Activity*** oder des *UML* ***Events*** bzw. *BPMN* ***Events*** mit dem eingefügten EPK ***Parallel Mege Gateway***.

Wird eine EPK in ein *UML* Aktivitätsdiagramm oder ein *BPMN* Diagramm überführt, in der ein EPK ***Parallel Merge Connector*** im Ausführungsfluss direkt hinter einer EPK ***Function*** oder einem EPK ***Event*** steht, dann wird dieser EPK ***Parallel Merge Connector*** bei der Überführung in die *UML*-Notation bzw. *BPMN*-Notation weggelassen. Die *UML* ***Activity Edge*** bzw. *BPMN* ***Sequence Flow*** Abbilder der EPK ***Control Flows***, die aus dem EPK ***Parallel Merge Connector*** ausgehen, laufen direkt aus dem *UML* ***Activity Node*** bzw. *BPMN* ***Activity*** Abbild der EPK ***Function*** oder dem *UML* ***Event*** bzw. *BPMN* ***Event*** Abbild des EPK ***Events***. Der EPK ***Control Flow*** zwischen EPK Function oder EPK Event und EPK ***Parallel Merge Connector*** wird nicht im *UML Aktivitätsdiagramm* bzw. *BPMN* Diagramm abgebildet (siehe Abb. 7.19).

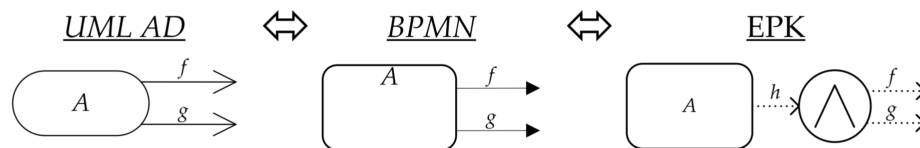


Abbildung 7.19.: Überführung eines *UML* ***Activity Nodes***, einer *BPMN* ***Activity***, bzw. einer EPK ***Function*** mit mehreren ausgehenden Flüssen in die jeweils anderen Notationen. Die Überführung eines *UML* ***Events***, eines *BPMN* ***Events***, bzw. eines EPK ***Events*** hat die gleichen Züge.

3. **Problemstellung**: In Anlehnung an die ersten beiden Problemstellungen, stellen sich Fragen zur Abbildung von *UML*, *BPMN*/ und EPK Diagrammen mit Notationselementen welche mehrere eingehende bzw. ausgehende Ausführungsflüsse besitzen.

   Lösung:

   a) Die Festlegung ob mehrere, in ***Activities*** oder ***Events*** eingehende ***Flows*** in Referenzdiagrammen eine *AND*- oder eine *OR*-Verknüpfung darstellen sollen, ist willkürlich. Weil es in *UML* Aktivitätsdiagrammen kein direkt vergleichbares Gegenstück zu *BPMN* ***Inclusive Branch Gateways*** bzw. EPK ***Inclusive Branch Connectors*** gibt und auf die Abbildung dieses Elements sowie dessen, den Ausführungsfluss inklusiv-zusammenführendes und nur in bestimmten Situationen in der EPK-Notation zwingend benötigten (siehe Gebrauch des EPK ***Inclusive Merge Connectors*** in Problemstellung 1.) Gegenstücks, in Geschäftsprozessmodellen verzichtet werden soll, soll auch in der Referenzsprache auf ***Inclusive Merge Connectors*** verzichtet werden. Weil mehrere eingehende Flüsse in ein *UML* ***Flow Final Node***, ein *UML* ***Activity Final Node*** sowie *BPMN* ***Activities*** und *BPMN* ***Events*** bei der Abbildung in ein Referenzdiagramm die Verwendung eines nicht vorhandenen ***Inclusive Merge Connector*** Elements benötigen würden, **stellen**

mehrere in eine *Activity* oder ein *Event* eingehende *Flows* in Referenzdiagrammen eine *OR*-Verknüpfungen dar. Hierdurch ergeben sich folgende Transformationsregeln:

i. Wird ein *UML* Aktivitätsdiagramm in ein Referenzdiagramm überführt, werden die eingehenden *UML* ***Activity Edges*** eines *UML* ***Activity Nodes*** oder *UML* ***Signal Events*** in entsprechende ***Flow*** Abbilder transformiert und über einen ***Parallel Merge Connector*** zusammengeführt. Ein zusätzlicher ***Flow*** verbindet den ***Parallel Merge Connector*** mit dem entsprechenden ***Activity*** Abbild des *UML* ***Activity Nodes*** oder dem ***Signal Event*** Abbild des *UML* ***Signal Events***.

ii. Da mehrere eingehende *UML* ***Activity Edges*** in *UML* ***Flow Final Nodes*** oder *UML* ***Activity Final Nodes*** wie in der Referenz eine *OR*-Verknüpfung darstellen, ist der Einsatz eines ***Connectors*** bei der Transformation in ein Referenzmodell unnötig. Stattdessen werden die ***Flow*** Abbilder der *UML* ***Activity Edges*** direkt an das ***Flow Final Event*** Abbild des *UML* ***Flow Final Nodes*** oder das ***Process Final Event*** Abbild des *UML* ***Activity Final Nodes*** geführt.

iii. Da mehrere eingehende *BPMN* ***Sequence Flows*** in *BPMN* ***Activities*** oder *BPMN* ***Events*** wie in der Referenz eine *OR*-Verknüpfung darstellen, ist der Einsatz eines ***Connectors*** bei der Transformation in ein Referenzmodell unnötig. Stattdessen werden die ***Flow*** Abbilder der *BPMN* ***Sequence Flows*** direkt an das ***Activity*** Abbild der *BPMN* ***Activity*** oder das ***Event*** Abbild des *BPMN* ***Events*** geführt.

iv. Wird eine EPK auf ein Referenzdiagramm abgebildet, in der ein EPK ***Inclusive Merge Connector*** im Ausführungsfluss direkt vor einer EPK ***Function*** oder einem EPK ***Event*** steht, dann wird auf die Abbildung dieses EPK ***Inclusive Merge Connectors*** bei der Überführung in das Referenzdiagramm verzichtet. Die ***Flow*** Abbilder der EPK ***Control Flows***, die in diesen ***Inclusive Merge Connector*** eingehen, laufen direkt in das ***Activity*** Abbild der EPK ***Function*** oder das ***Event*** Abbild des EPK ***Events***. Der EPK ***Control Flow*** zwischen EPK ***Inclusive Merge Connector*** und EPK ***Function*** oder EPK ***Event*** wird nicht im Referenzdiagramm abgebildet.

b) Da sowohl die UML- als auch die BPMN-Notation die konjunktiven Aufteilung des Ausführungsflusses auf mehrere unabhängige Zweige nach der Ausführung einer Tätigkeit oder eines Ereignisses vorschreiben und die Abbildung dieser Gegebenheiten in einer EPK durch den Einsatz eines EPK Parallel Branch Connectors möglich ist, **stellen mehrere in aus einer Activity oder einem Event ausgehende Flows in Referenzdiagrammen eine *AND*-Verzweigung dar**. Hierdurch ergeben sich folgende Transformationsregeln:

i. Bei der Abbildung eines *UML* Aktivitätsdiagrammes mit mehreren aus einem *UML* ***Activity Node*** oder *UML* ***Event*** ausgehenden *UML* ***Activity Edges*** in ein Referenzdiagramm, laufen die ***Flow*** Abbilder der *UML* ***Activity Edges*** direkt aus dem ***Activity*** Abbild des *UML* ***Activity Nodes*** oder aus dem ***Event*** Abbild des *UML* ***Events***.

ii. Bei der Abbildung eines *BPMN* Diagrammes mit mehreren aus einer *BPMN* ***Activity*** oder eines *BPMN* ***Events*** ausgehenden *BPMN* ***Sequence Flows*** in ein Referenzdiagramm, laufen die ***Flow*** Abbilder der *BPMN* ***Sequence Flows*** direkt aus dem ***Activity*** Abbild der *BPMN* ***Activity*** oder aus dem ***Event*** Abbild des *BPMN* ***Events***.

iii. Wird eine EPK auf ein Referenzdiagramm abgebildet, in der ein EPK ***Parallel Merge Connector*** im Ausführungsfluss direkt hinter einer EPK ***Function*** oder einem EPK ***Event*** steht, dann wird auf die Abbildung dieses EPK ***Parallel Merge Connectors*** bei der Überführung in das Referenzdiagramm verzichtet. Die ***Flow*** Abbilder der EPK ***Control Flows*** die aus diesem ***Parallel Merge Connector*** kommen, laufen direkt aus dem ***Activity*** Abbild der EPK ***Function*** oder dem ***Event*** Abbild des EPK ***Events***. Der EPK ***Control Flow*** zwischen EPK ***Function*** oder EPK ***Event*** und EPK ***Parallel Merge Connector*** wird nicht im Referenzdiagramm abgebildet.

4. **Problemstellung**: Da *UML* ***Activity Edges*** und *BPMN* ***Sequence Flows***, im Gegensatz zu EPK ***Control Flows***, einen Bezeichner und eine Ausführungsbedingung – *UML* ***Guard***, *BPMN* ***Condition*** – aufweisen können und die Rolle der Ausführungsbedingung in EPK durch die EPK ***Intermediate Events*** übernommen wird, stellt sich die Frage, wie ein *UML* Aktivitätsdiagramm oder *BPMN* Diagramm mit diesen Merkmalen auf eine EPK abgebildet werden soll. Auch die umkehrte Abbildung von einer EPK mit diesen Merkmalen auf ein *UML* Aktivitätsdiagramm oder ein *BPMN* Diagramm abgebildet wirft Fragen auf. Das Vorhaben wird zusätzlich dadurch erschwert, dass sich EPK ***Events*** und EPK ***Functions*** im Ausführungsfluss einer EPK ablösen müssen. Letztlich muss auch geklärt werden, wie Bezeichner und Ausführungsbedingungen im Referenzmodell gespeichert werden sollen.

   Lösung: Da die Ausführungsbedingungen in einer EPK als EPK ***Event*** dargestellt werden, liegt es nahe, *UML* ***Guards*** bzw. *BPMN* ***Conditions*** in EPK ***Events*** zu überführen. Zusätzlich können auch die Kantenbezeichner im EPK ***Event*** gespeichert werden. Zur besseren Unterscheidung steht die Ausführungsbezeichnung in eckigen Klammern hinter dem Bezeichner im EPK ***Event*** (siehe Abb. 7.20 und Abb. 7.21).

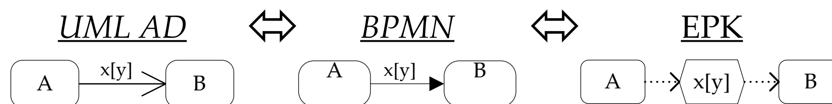


Abbildung 7.20.: Abbildung einer *UML* ***Activity Edge*** mit Bezeichner und ***Guard*** bzw. eines *BPMN* ***Sequence Flows*** mit Bezeichner und ***Condition*** in EPK-Notation.

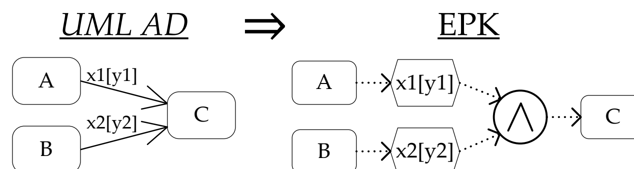


Abbildung 7.21.: Abbildung mehrerer *UML* ***Activity Edges*** mit Bezeichnern und ***Guards*** in EPK-Notation.

Verbindet ein *UML* ***Activity Edge*** ohne Ausführungsbedingung zwei *UML* ***Activity Nodes*** oder ein *BPMN* ***Sequence Flow*** ohne Ausführungsbedingung zwei *BPMN* ***Activities***, wird bei der Überführung in die EPK-Notation ein EPK ***Event*** mit der Ausführungsbedingung

true zwischen die EPK ***Function*** Abbilder der *UML* ***Activity Nodes*** bzw. der *BPMN* ***Activities*** geschaltet (siehe Abb. 7.22). Der boolsche Wert *true* ergibt sich aus dem Standardwert für Ausführungsbedingungen von *UML* ***Activity Edges***.

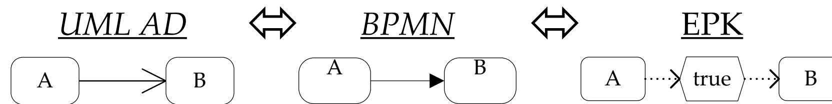


Abbildung 7.22.: Abbildung einer *UML* ***Activity Edge*** ohne Bezeichner und ***Guard*** bzw. eines *BPMN* ***Sequence Flows*** ohne Bezeichner und ***Condition*** in EPK-Notation

Ist ein *UML* ***Activity Node*** bzw. eine *BPMN* ***Activity*** oder ein *UML* ***Event*** bzw. *BPMN* ***Event*** mit einem Signal- oder Endereignis über eine Kante mit Bezeichner und Ausführungsbedingung verbunden, wird der Name des Signal- oder Endereignisses, getrennt durch ein Semikolon, vor dem Kantenbezeichner und der Ausführungsbedingung notiert (siehe Abb. 7.23).

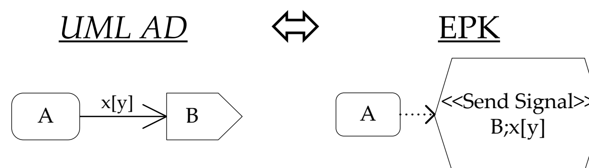


Abbildung 7.23.: Abbildung einer *UML* ***Activity Edge*** mit Bezeichner und ***Guard***, welche ein *UML* ***Activity Node*** mit einer ***Send Signal Action*** verbindet, in EPK-Notation.

Weist eine EPK mehrere EPK ***Functions*** auf, welche über einen EPK ***Connector*** und ein EPK ***Event*** mit einer weiteren EPK ***Function*** verbunden ist, dann enthält das EPK ***Event*** die Bezeichner und Ausführungsbedingungen aller in den EPK ***Connector*** eingehenden EPK ***Control Flows***. Der Bezeichner ist dabei durch Kommata von der Ausführungsbedingung getrennt (siehe Abb. 7.24). Das erste Bezeichner-Ausführungsbedingung-Paar stellt bei horizontaler Ausrichtung der EPK ***Control Flows*** am EPK ***Connector*** den Bezeichner und die Ausführungsbedingung des obersten EPK ***Control Flows*** dar, bei vertikaler Ausrichtung die des am weitesten links gelegenen EPK ***Control Flows***. Besitzt der EPK ***Event*** weniger Bezeichner-Ausführungbedingung-Paare als der EPK ***Connector*** eingehende EPK ***Control Flows*** hat, so wird das letzte Paar dem dazugehörigen und allen weiteren EPK ***Control Flows*** zugeteilt (siehe Abb. 7.25). Ein EPK ***Event*** darf nicht mehr Bezeichner-Ausführungbedingung-Paare besitzen, als der vorangehende EPK ***Connector*** eingehende EPK ***Control Flows*** besitzt (siehe Abb. 7.26).

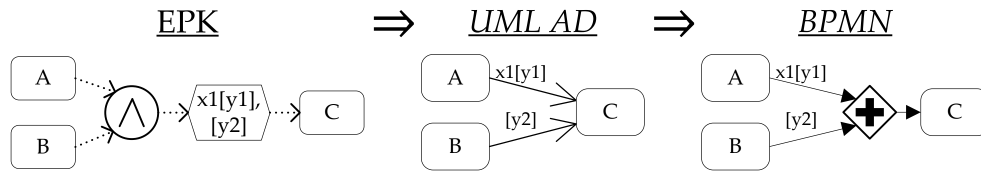


Abbildung 7.24.: Abbildung eines EPK ***Events*** mit mehreren Bezeichnern und Ausführungsbedingungen in *UML*- bzw. *BPMN*-Notation.

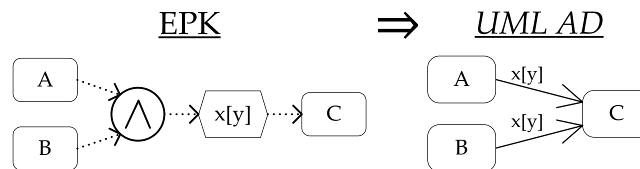


Abbildung 7.25.: Abbildung eines EPK ***Events***, mit weniger Bezeichner-Ausführungs-Paaren als der EPK ***Parallel Merge Connector*** eingehende EPK ***Control Flows*** besitzt, in *UML*-Notation.

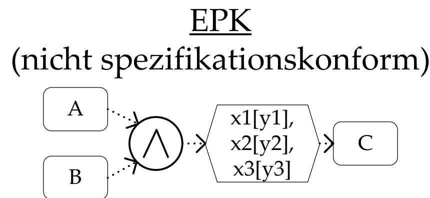


Abbildung 7.26.: Die Abbildung eines EPK ***Events***, mit mehr Bezeichner-Ausführungs-Paaren als der EPK ***Parallel Merge Connector*** eingehende EPK ***Control Flows*** besitzt, ist nicht möglich.

Obwohl die Bezeichner-Bedingungs-Paare von EPK ***Control Flows*** graphisch in den darauffolgenden EPK ***Events*** aufgeschrieben werden, lassen sich die Paare den passenden EPK ***Control Flow*** zuteilen. In der nicht-graphischen Beschreibung einer EPK erlaubt dies die direkte Zuweisung der Bezeichner und Bedingungen als Parameter von EPK ***Control Flows*** (siehe Unterkapitel 8.3.2).

Im Referenzmodell besitzen ***Flows*** separate Attribute für den Bezeichner und die Ausführungsbedingung. Der Standardwert der Ausführungsbedingung ist der boolsche Wert *true*.

7.2.2.1. Resource Flow

Vergleichbar zu:

- *UML* ***Object Flow*** (siehe Unterkapitel 3.3.2.1) und

- *BPMN* ***Sequence Flow*** mit transportierten ***Data Objects*** (siehe Unterkapitel 4.3.2.1 und Unterkapitel 4.3.3), während
- EPK kein direktes Gegenstück besitzen (siehe Paragraph Unterschiede, Problemstellung 1).

Beschreibung:

Resource Flows sind eine Spezialisierung von ***Flows*** und modellieren zusätzlich zum Ausführungsfluss auch den Transport von Daten im Diagramm. ***Activities*** verarbeiten und produzieren Datensätze (***Resources***), welche ihnen über ***Resource Flows*** zugeführt und abgeführt werden können.

Unterschiede:

Die in den verschiedenen Notationssprachen unterschiedlichen Implementierungen des Umgangs, der Repräsentation und der Weitergabe von Datensätzen werfen einige Fragen auf:

1. **Problemstellung**: Die *UML* Aktivitätsdiagramme bieten mit den *UML* ***Object Flows*** ein direktes Gegenstück für die ***Resource Flows***. In der BPMN lässt sich das Verhalten von ***Resource Flows*** mit *BPMN* ***Sequence Flows***, welche *BPMN* ***Data Objects*** transportieren, nachstellen. Einzig die EPK bietet kein direkt vergleichbares Gegenstück. Statt EPK ***Information Objects*** an EPK ***Control Flows*** binden zu können, werden sie über gerichtete Kanten (*Information Flow*) direkt mit den EPK ***Functions*** verbunden, welche die Datensätze als Eingabe für die Ausführung benötigen und/ oder Datensätze produzieren und als Ausgabedaten an andere EPK ***Functions*** weitergeben. Wie sieht eine korrekte Abbildung von ***Resource Flows*** in EPK und umgekehrt, wie sieht der Datenaustausch aus EPK in anderen Sprachen und in der Referenz aus?

   Lösung: Der Datenaustausch eines *UML* ***Objects*** x zwischen zwei *UML* ***Activity Nodes*** und/ oder *UML* ***Events*** A und B über einen *UML* ***Object Flow*** f in einem *UML* Aktivitätsdiagramm, lässt sich auch als Ausgabe des Datensatzes x durch A und als Eingabe von x durch B verstehen, während f nur die Ausführungsreihenfolge festlegt und für den Transport von x zuständig ist. Die gleiche Beobachtung kann in *BPMN* Diagrammen gemacht werden. Dementsprechend kann der Datenaustausch aus *UML* Aktivitätsdiagrammen oder *BPMN* Diagrammen in EPK abgebildet werden, indem die Abbilder von A und B über EPK ***Control Flow*** Abbilder verbunden werden und das EPK ***Information Object*** Abbild von x als Ausgabe von A und als Eingabe von B notiert wird (siehe Abb. 7.27).

   EPK ***Information Objects*** lassen sich also indirekt den für ihren Transport im Ausführungsfluss zuständigen EPK ***Control Flows*** zuordnen.

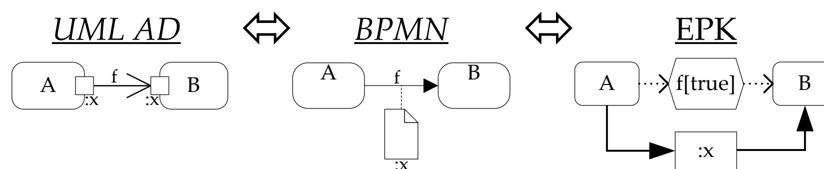


Abbildung 7.27.: Darstellung des Datenaustausches in den verschiedenen Notationssprachen.

2. **Problemstellung**: Der Ausführungszeitpunkt bei einem gemischten Einsatz von eingehenden *UML* ***Control Flows*** und *UML* ***Object Flows*** bzw. *BPMN* ***Sequence Flows*** und *BPMN* ***Sequence Flows*** welche *BPMN* ***Data Objects*** transportieren, unterscheidet sich nicht von dem in Unterkapitel 7.2.2 für mehrere eingehende *UML* ***Control Flows*** bzw. *BPMN* ***Sequence Flows*** beschriebenem Zeitpunkt.

   Demnach führt die Überführung einer *BPMN* ***Activity*** oder eines *BPMN* ***Signal Events*** mit einer Kombination aus mehreren eingehenden *BPMN* ***Sequence Flows*** und *BPMN* ***Sequence Flows*** welche *BPMN* ***Data Objects*** transportieren, in die *UML*-Notation, zur Zusammenführung von entsprechenden *UML* ***Control Flow*** und *UML* ***Object Flows*** Abbildern über ein *UML* ***Join Node*** mit inklusivem ***Join Specification***-Ausdruck (siehe Abb. 7.28). Die *UML*-Spezifikation verbietet indes ausdrücklich den gemischten Einsatz von *UML* ***Control Flows*** und *UML* ***Object Flows*** an *UML* ***Control Nodes***. Eine solche Einschränkung existiert nicht in den *BPMN*- und EPK-Spezifikationen.

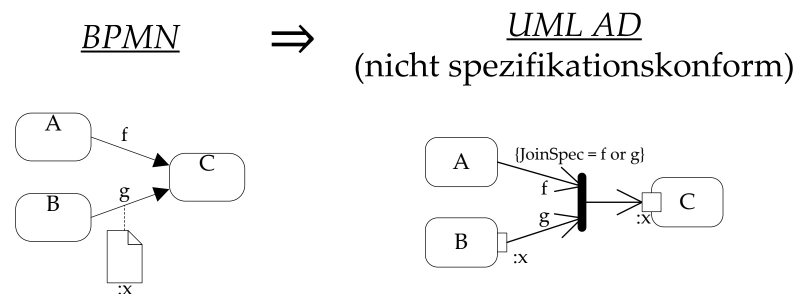


Abbildung 7.28.: Die *UML*-Spezifikation verbietet ausdrücklich den gemischten Einsatz von *UML* ***Control Flows*** und *UML* ***Object Flows*** an *UML* ***Control Nodes***, wie er bei der Abbildung dieses *BPMN* Diagrammauszugs in *UML*-Notation entsteht.

   Lösung: In *BPMN* Diagrammen müssen alle in *BPMN* ***Activities*** oder *BPMN* ***Signal Events*** eingehende *BPMN* ***Sequence Flows*** entweder alle *BPMN* ***Data Objects*** transportieren oder gar keine (siehe Abb. 7.29), damit bei der Überführung kein *UML* ***Join Node*** mit eingehenden *UML* ***Control Flows*** und *UML* ***Object Flows*** hervorgeht.

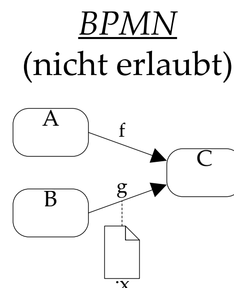


Abbildung 7.29.: Aus Kompatibilitätsgründen zur *UML*-Notation ist die Modellierung eines BPMN Diagrammauszugs nach diesem Schema nicht erlaubt.

Aus Kompatibilitätsgründen zur *UML*-Notation müssen die ein- und ausgehenden *BPMN* ***Sequence Flows*** von *BPMN* ***Gateways*** und EPK ***Control Flows*** von EPK ***Connectors*** entweder alle Daten transportieren (entspricht *UML* ***Object Flow***), oder gar keine (entspricht *UML* ***Control Flow***).

Als einzige Ausnahme zu dieser Regel dürfen *BPMN* ***Parallel Merge Gateways*** und EPK ***Parallel Merge Connectors***, welche im Ausführungsfluss über genau eine ausgehende Kante direkt mit einer *BPMN* ***Activity*** oder einem *BPMN* ***Signal Event*** bzw. einer EPK ***Function*** oder einem EPK ***Signal Event***, eine Mischung ein- und ausgehender *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** besitzen, von denen welche Daten transportieren und andere keine Daten transportieren. Diese Ausnahme ermöglicht die korrekte Abbildung einer Kombination mehrerer in einen *UML* ***Activity Node*** oder ein *UML* ***Signal Event*** eingehender *UML* ***Control Flows*** und *UML* ***Object Flows*** in *BPMN*- und EPK-Notation (siehe Abb. 7.30).

Die gleichen Einschränkungen gelten für ***Connectors*** im Referenzmodell.

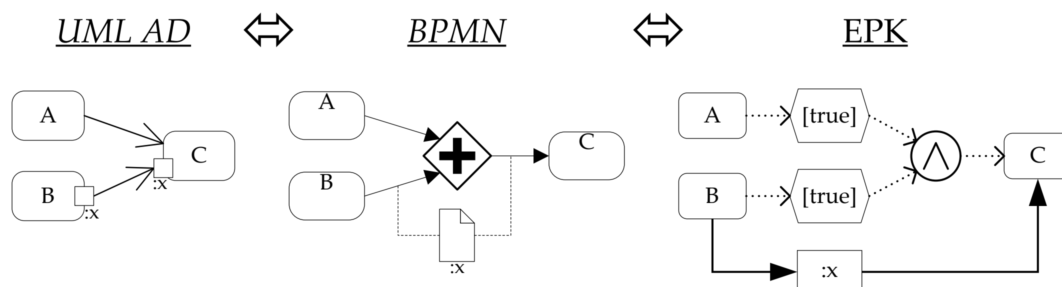


Abbildung 7.30.: Um die Kompatibilität zur *UML*-Notation zu wahren, dürfen *BPMN* ***Parallel Merge Gateways*** und EPK ***Parallel Merge Gateways***, welche direkt mit einem *BPMN* ***Activity Node*** oder einem *BPMN* ***Signal Event*** bzw. einer EPK ***Function*** oder einem EPK ***Signal Event*** verbunden sind, sowohl *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** die Daten transportieren oder die keine Daten transportieren, haben.

3. **Probemstellung**: Die *UML*-Spezifikation schreibt für *UML* ***Initial Node*** vor, dass alle ausgehenden *UML* ***Activity Edges*** vom Untertyp *UML* ***Control Flow*** sein müssen (vgl.**[OMG09b]**). Die *BPMN*- und EPK-Spezifikationen schreiben diesbezüglich keine Einschränkungen vor.

   Lösung: In Anlehnung an die *UML*-Notation dürfen die aus *BPMN* ***Start Events*** bzw. EPK ***Start Events*** ausgehenden *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** keine Daten transportieren. Gleiches gilt in Referenzmodellen, in denen alle aus ***Start Events*** ausgehende Kanten vom Typ ***Flow*** sein müssen.

4. **Problemstellung**: Die *UML*-Spezifikation schreibt für *UML* ***Flow Final Node*** und *UML* ***Activity Final Node*** vor, dass alle eingehenden *UML* ***Activity Edges*** vom Untertyp *UML* ***Control Flow*** sein müssen (vgl. **[OMG09b]**). Die *BPMN*- und EPK-Spezifikationen schreiben diesbezüglich keine Einschränkungen vor.

   Lösung: In Anlehnung an die *UML*-Notation dürfen die in *BPMN* ***End Events*** oder *BPMN* ***Terminate End Events*** bzw. EPK ***End Events*** oder EPK ***Terminate End Events*** eingehenden

BPMN ***Sequence Flows*** bzw. EPK ***Control Flows*** keine Daten transportieren. Gleiches gilt in Referenzmodellen, in denen alle aus ***End Events*** oder ***Terminate End Events*** ausgehende Kanten vom Typ ***Flow*** sein müssen.

7.2.3. Connectors

Vergleichbar zu:

- *UML* ***Control Nodes*** (siehe Unterkapitel 3.3.2.3),
- *BPMN* ***Gateways*** (siehe Unterkapitel 4.3.2.3) und
- EPK ***Connectors*** (siehe Unterkapitel 5.3.2.3).

Beschreibung:

Connectors sind für die Verzweigung (***Branch***) und Zusammenführung (***Merge***) von ***Flows*** und ***Resource Flows*** zuständig und koordinieren den Ausführungsfluss eines Diagrammes.

Bei ***Connectors*** wird unterschieden zwischen:

- ***Exclusive Branch Connectors*** (siehe Unterkapitel 7.2.3.1),
- ***Parallel Branch Connectors*** (siehe Unterkapitel 7.2.3.2),
- ***Exclusive Merge Connectors*** (siehe Unterkapitel 7.2.3.3) und
- ***Parallel Merge Connectors*** (siehe Unterkapitel 7.2.3.4).

Unterschiede:

UML ***Control Nodes***, *BPMN* ***Gateways*** und EPK ***Connectors*** erfüllen die gleiche Aufgaben und unterscheiden sich nur leicht:

1. **Problemstellung**: Die *UML*-Spezifikaiton schreibt vor, dass die ein- und ausgehenden *UML* ***Activity Edges*** von *UML* ***Control Nodes*** alle vom gleichen Untertyp, also entweder alle *UML* ***Control Flows*** oder alle *UML* ***Object Flows***, sein müssen. Die BPMN- und UML-Spezifikation machen hier jedoch keinen Unterschied.

   Lösung: Aus Kompatibilitätsgründen zur *UML*-Notation müssen die ein- und ausgehenden *BPMN* ***Sequence Flows*** von *BPMN* ***Gateways*** und EPK ***Control Flows*** von EPK ***Connectors*** entweder alle Daten transportieren (entspricht *UML* ***Object Flow***), oder gar keine (entspricht *UML* ***Control Flow***).

   Als einzige Ausnahme zu dieser Regel dürfen *BPMN* ***Parallel Merge Gateways*** und EPK ***Parallel Merge Connectors***, welche im Ausführungsfluss über genau eine ausgehende Kante direkt mit einer *BPMN* ***Activity*** oder einem *BPMN* ***Signal Event*** bzw. einer EPK ***Function*** oder einem EPK ***Signal Event***, eine Mischung ein- und ausgehender *BPMN* ***Sequence Flows*** bzw. EPK ***Control Flows*** besitzen, von denen welche Daten transportieren und andere keine Daten transportieren, haben. Diese Ausnahme ermöglicht die korrekte Abbildung einer Kombination mehrerer in einen *UML* ***Activity Node*** oder ein *UML* ***Signal Event*** eingehender *UML* ***Control Flows*** und *UML* ***Object Flows*** in *BPMN*- und EPK-Notation (siehe Abb. 7.30).

   Die gleichen Einschränkungen gelten für ***Connectors*** im Referenzmodell.

7.2.3.1. Exclusive Branch Connector

Vergleichbar zu:

- *UML* ***Decision Node*** (siehe Unterkapitel 3.3.2.3),
- *BPMN* ***Data-based Exclusive Branch Gateway*** (siehe Unterkapitel 4.3.2.3) und
- EPK ***Exclusive Branch Connector*** (siehe Unterkapitel 5.3.2.3).

Beschreibung:

Exclusive Branch Connectors und ***Exclusive Merge Connectors*** dienen der Modellierung alternativer Pfade – *XOR*-Verknüpfung. An einem ***Exclusive Branch Connector*** wird der eingehende Ausführungsfluss anhand der Auswertung disjunkter Ausführungsbedingungen an den ausgehenden ***Flows*** oder ***Resource Flows*** auf genau einen Pfad gelenkt.

Unterschiede:

UML ***Decision Nodes***, *BPMN* ***Data-based Exclusive Branch Gateways*** und EPK ***Exclusive Branch Connectors*** haben die gleiche Bedeutung in allen drei Sprachen. Es bleibt anzudeuten, dass die Ausführungsbedingungen/ Entscheidungsgrundlage(n) bei der Modellierung in *UML* als *UML* ***Guards*** an die ausgehenden *UML* ***Activity Edges*** statt direkt am *UML* ***Decision Node*** angebracht werden sollten (vgl. Notationsmöglichkeiten in [OMG09b]). In *BPMN* sollten die Ausdrücke der *BPMN* ***Conditions*** an *BPMN* ***Sequence Flows***, entgegen der *BPMN*-Spezifikation, stets disjunkt sein. Gleiches gilt für die EPK-Notation, bei der die Ausführungsbedingungen an den einzelnen Ausführungszweigen als EPK ***Intermediate Event*** stehen. Im Referenzmodell werden die disjunkten Ausführungsbedingungen an die ***Flows*** oder ***Resource Flows*** geschrieben.

7.2.3.2. Parallel Branch Connector

Vergleichbar zu:

- *UML* ***Fork Node*** (siehe Unterkapitel 3.3.2.3),
- *BPMN* ***Parallel Branch Gateway*** (siehe Unterkapitel 4.3.2.3) und
- EPK ***Parallel Branch Connector*** (siehe Unterkapitel 5.3.2.3).

Beschreibung:

Parallel Branch Connectors sind das Gegenstück zu ***Parallel Merge Connectors. Parallel Branch Connectors*** teilen den Ausführungsfluss in mehrere nebenläufige Flüsse, die unabhängig voneinander abgearbeitet werden.

Unterschiede:

UML ***Fork Nodes***, *BPMN* ***Parallel Branch Gateways*** und EPK ***Parallel Branch Connectors*** erfüllen die gleiche Rolle und unterscheiden sich nur graphisch voneinander.

7.2.3.3. Exclusive Merge Connector

Vergleichbar zu:

- *UML **Merge Node*** (siehe Unterkapitel 3.3.2.3),
- *BPMN **Data-based Exclusive Merge Gateway*** (siehe Unterkapitel 4.3.2.3) und
- EPK ***Exclusive Merge Connector*** (siehe Unterkapitel 5.3.2.3).

Beschreibung:
Exclusive Merge Connectors und ***Exclusive Branch Connectors*** dienen der Modellierung alternativer Pfade – *XOR*-Verknüpfung. An ***Exclusive Merge Connectors*** fließt der Ausführungsfluss über genau einen der eingehenden Pfade/ ***Flows*** oder ***Resource Flows*** ein.

Unterschiede:
*UML **Merge Nodes**, BPMN **Data-based Exclusive Merge Gateways*** und EPK ***Exclusive Merge Connectors*** erfüllen die gleiche Rolle und unterscheiden sich nur graphisch voneinander.

7.2.3.4. Parallel Merge Connector

Vergleichbar zu:

- *UML **Join Node*** (siehe Unterkapitel 3.3.2.3),
- *BPMN **Parallel Merge Gateway*** (siehe Unterkapitel 4.3.2.3) und
- EPK ***Parallel Merge Connector*** (siehe Unterkapitel 5.3.2.3).

Beschreibung:
Parallel Merge Connectors sind das Gegenstück zu ***Parallel Branch Connectors. Parallel Merge Connectors*** synchronisieren mehrere eingehende ***Flows*** oder ***Resource Flows***, d.h. sie geben den sie verlassenden Ausführungsfluss erst frei, nachdem alle in sie eingehende Flüsse anliegen – *AND*-Verknüpfung.

Unterschiede:
*UML **Join Nodes**, BPMN **Parallel Merge Gateways*** und EPK ***Parallel Merge Connectors*** erfüllen die gleiche Rolle und unterscheiden sich nur graphisch voneinander.

7.3. Resources

Vergleichbar zu:

- *UML **Objects*** (siehe Unterkapitel 3.3.3),
- *BPMN **Data Objects*** (siehe Unterkapitel 4.3.2.3) und
- EPK ***Information Objects*** (siehe Unterkapitel 5.3.3).

Beschreibung:

Resources stellen die bei der Bearbeitung des Geschäftsprozesses von ***Activities*** verwendeten und produzierten Daten dar. ***Resources*** können einem Geschäftsprozess auch von außerhalb zugeführt und nach außen abgeführt werden.

Unterschiede:

1. **Problemstellung**: In *UML* Aktivitätsdiagrammen können *UML* ***Objects*** indirekt über *UML* ***Object Nodes*** zwischen *UML* ***Activity Nodes*** in verschiedenen *UML* ***Activities*** und/ oder *UML* ***Activity Diagrams*** ausgetauscht werden (siehe Unterkapitel 3.3.3 und 3.3.1.2; siehe Abb. 7.31). Die *BPMN* und EPK bieten hierzu kein direktes Gegenstück. Neben der Frage wie dieser Datenaustausch in *BPMN* und EPK Diagrammen dargestellt werden soll, stellt sich zusätzlich die Frage nach der korrekten Darstellung im Referenzmodell.

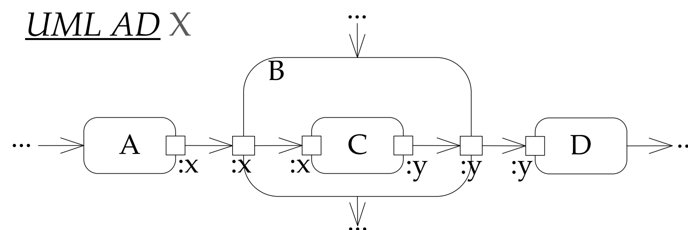


Abbildung 7.31.: Datenaustausch zwischen zwei *UML* ***Activities***/ *UML* ***Activity Diagrams***.

Lösung: Die Lösung setzt sich aus Teillösungen für die verschiedenen Notationen zusammen:

a) In der *BPMN*-Notation werden Spezialisierung von *BPMN* ***Signal Throwing Events*** und *BPMN* ***Signal Catching Events*** eingeführt, die *BPMN* ***Resource Throwing Events*** und *BPMN* ***Resource Catching Events***, deren Bezeichner am Anfang jeweils um den Stereotypen ***<<Resource Throwing Event>>*** bzw. ***<<Resource Catching Event>>*** erweitert werden. Um die visuelle Notation zu entlasten, können die Elemente anstatt mit Steoreotypen auch mit einem #-Symbol am Ende des Bezeichners gekennzeichnet werden. *BPMN* ***Resource Throwing Events*** stehen immer am Ende eines Ausführungsflusses, beenden nur die in sie eingehenden Ausführungsflüsse und übertragen alle eingehenden *BPMN* ***Data Objects*** an alle *BPMN* ***Resource Catching Events*** mit gleichem Namen. *BPMN* ***Resource Catching Events*** geben alle erhaltenen *BPMN* ***Data Objects*** über alle ausgehenden Ausführungsflüsse weiter und können als Alternative zu *BPMN* ***Start Events*** in Diagrammen eingesetzt werden (siehe Abb. 7.32).

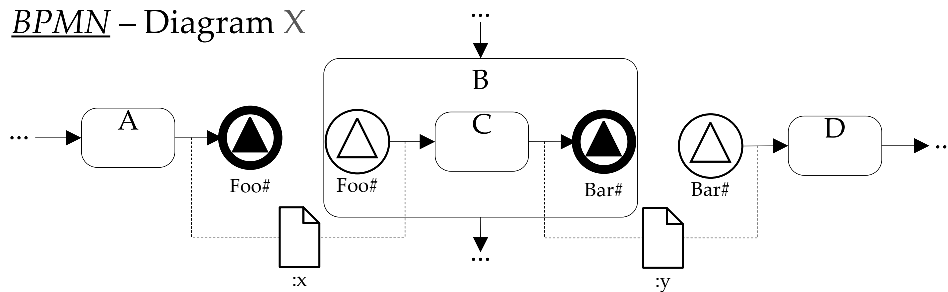


Abbildung 7.32.: Datenaustausch zwischen zwei *BPMN* ***Sub-Processes***/ *BPMN* Diagrammen anhand der neu eingeführten *BPMN* ***Resource Throwing Events*** und *BPMN* ***Resource Catching Events***. Um die Übersichtlichkeit der visuellen Notation zu erhöhen, wurden die Bezeichner der *BPMN* ***Resource Throwing Events*** und *BPMN* ***Resource Catching Events*** mit einem #-Symbol statt mit Stereotypen ausgestattet.

b) In der EPK-Notation werden Spezialisierung von EPK ***Signal Throwing Events*** und EPK ***Signal Catching Events*** eingeführt, die EPK ***Resource Throwing Events*** und EPK ***Resource Catching Events***, deren Bezeichner am Anfang jeweils um den Stereotypen **<<*Resource Throwing Event*>>** bzw. **<<*Resource Catching Event*>>** erweitert werden. Um die visuelle Notation zu entlasten können die Elemente alternativ anstatt mit Steoreotypen auch mit einem #-Symbol am Ende des Bezeichners gekennzeichnet werden. EPK ***Resource Throwing Events*** stehen immer am Ende eines Ausführungsflusses, beenden nur die in sie eingehenden Ausführungsflüsse und übertragen alle eingehenden EPK ***Information Objects*** an alle EPK ***Resource Catching Events*** mit gleichem Namen. EPK ***Resource Catching Events*** geben alle erhaltenen EPK ***Information Objects*** über alle ausgehenden Ausführungsflüsse weiter und können als Alternative zu EPK ***Start Events*** in EPK eingesetzt werden (siehe Abb. 7.33).

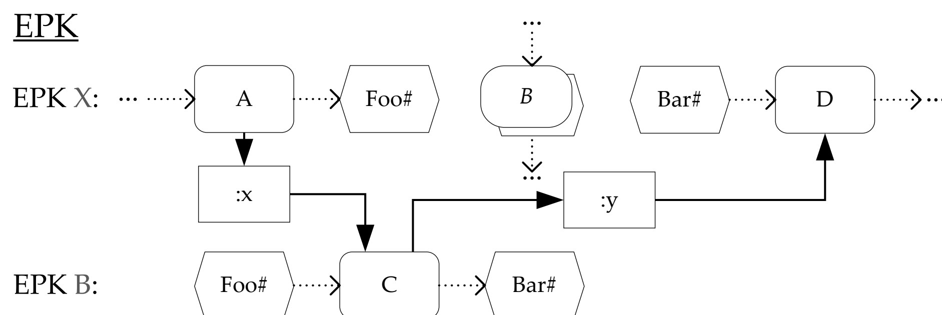


Abbildung 7.33.: Datenaustausch zwischen zwei EPK ***Complex Functions***/ EPK anhand der neu eingeführten EPK ***Resource Throwing Events*** und EPK ***Resource Catching Events***. Um die Übersichtlichkeit der visuellen Notation zu erhöhen, wurden die Bezeichner der EPK ***Resource Throwing Events*** und EPK ***Resource Catching Events*** mit einem #-Symbol statt mit Stereotypen ausgestattet.

c) An der nicht-graphischen *UML*-Notation zum Datenaustausch wird eine Anpassung durchgeführt. In der *UML*-Notation werden Spezialisierung von *UML **Send Signal Actions*** und *UML **Accept Signal Actions*** eingeführt, die *UML **Send Resource Actions*** und *UML **Accept Resource Actions***, deren Bezeichner am Anfang jeweils um den Stereotypen ***<<Send Signal Action>>*** bzw. ***<<Accept Signal Action>>*** erweitert werden. Um die visuelle Notation zu entlasten, können die Elemente anstatt mit Steoreotypen auch mit einem #-Symbol am Ende des Bezeichners gekennzeichnet werden. *UML **Send Signal Actions*** stehen immer am Ende eines Ausführungsflusses, beenden nur die in sie eingehenden Ausführungsflüsse und übertragen alle eingehenden *UML **Objects*** an alle *UML **Accept Signal Actions*** mit gleichem Namen. *UML **Accept Signal Actions*** geben alle erhaltenen *UML **Objects*** über alle ausgehenden Ausführungsflüsse weiter und können als Alternative zu *UML **Initial Nodes*** in Diagrammen eingesetzt werden (siehe Abb. 7.34).

Besitzt eine *UML **Activity*** nach Wegfall aller eingehenden *UML **Object Flows*** keine weiteren eingehenden *UML **Activity Edges***, wird ein *UML **Initial Node*** mit generiertem Namen eingeführt und über einen ausgehende *UML **Control Flow*** direkt mit der *UML **Activity*** verbunden. Besitzt eine *UML **Activity*** nach Wegfall aller ausgehenden *UML **Object Flows*** keine weiteren ausgehenden *UML **Actvity Edges***, wird die *UML **Activity*** über einen generierten *UML **Control Flow*** direkt mit einem eingeführten *UML **Flow Final Node*** mit generiertem Namen verbunden. Durch diese Maßnahmen ist garantiert, dass eine *UML **Activity*** stets mindestens eine ein- und eine ausgehenden *UML **Activity Edge*** besitzt.

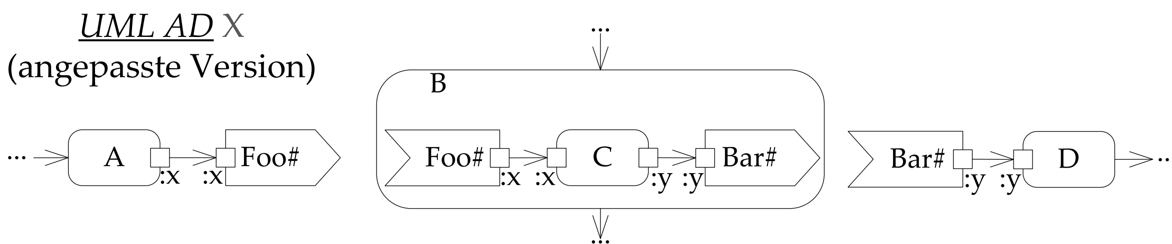


Abbildung 7.34.: Alternative, nicht-graphische Darstellung des Datenaustausch zwischen zwei *UML **Activities***/ *UML **Activity Diagrams*** nach der Konvertierung in ein, zu dem in dieser Arbeit erstellten, *UML*-metamodellkompatiblen Modell. Um die Übersichtlichkeit der visuellen Notation zu erhöhen, wurden die Bezeichner der *UML **Send Resource Actions*** und UML ***Accept Resource Action*** mit einem #-Symbol statt mit Stereotypen ausgestattet.

d) In der Referenz werden Spezialisierung von ***Signal Throwing Events*** und ***Signal Catching Events*** eingeführt, die ***Resource Throwing Events*** und ***Resource Catching Events***, deren Bezeichner am Anfang jeweils um den Stereotypen ***<<Resource Throwing Event>>*** bzw. ***<<Resource Catching Event>>*** erweitert werden. Um die visuelle Notation zu entlasten, können die Elemente anstatt mit Steoreotypen auch mit einem #-Symbol am Ende des Bezeichners gekennzeichnet werden. ***Resource Throwing Events*** stehen immer

am Ende eines Ausführungsflusses, beenden nur die in sie eingehenden Ausführungsflüsse und übertragen alle eingehenden ***Resources*** an alle ***Resource Catching Events*** mit gleichem Namen. ***Resource Catching Events*** geben alle erhaltenen ***Resources*** über alle ausgehenden Ausführungsflüsse weiter und können als Alternative zu ***Start Events*** in Diagrammen eingesetzt werden.

2. **Problemstellung**: Während *UML **Objects*** einen Instanznamen, einen Typnamen und einen Zustand besitzen, haben *BPMN **Data Objects*** nur einen Namen sowie einen Zustand und EPK ***Information Objects*** weisen sogar nur einen Namen auf.

   Lösung: Die Lösung setzt sich aus Teillösungen für die verschiedenen Notationen zusammen:

   a) Bei der Abbildung eines *UML **Objects*** mit Instanznamen, ***Object***-Typnamen und Zustand auf ein *BPMN **Data Object***, wird der Instanzname, gefolgt von einem Doppelpunkt und dem *UML **Object***-Typnamen, im Namen des *BPMN **Data Objects*** gespeichert. Beispiel: Ein *UML **Object*** mit Instanznamen *foo*, *UML **Object***-Typnamen *baz* und Status *bar* wird in ein *BPMN **Data Object*** mit Namen *foo:baz* und Zustand *bar* überführt.

   b) Bei der Abbildung eines *UML **Objects*** mit Instanznamen, ***Object***-Typnamen und Zustand auf ein EPK ***Information Object***, wird der Instanzname, gefolgt von einem Doppelpunkt und dem *UML **Object***-Typnamen sowie dem Status in eckigen Klammern, im Namen des *EPK **Information Objects*** gespeichert. Beispiel: Ein *UML **Object*** mit Instanznamen *foo*, *UML **Object***-Typnamen *baz* und Status *bar* wird in ein *EPK **Information Object*** mit Namen *foo:baz[bar]* überführt.

   c) Im Referenzdiagramm besitzen ***Resources*** separate Attribute für den Instanznamen, den Typnamen und den Zustand.

7.4. Participant Partitions

Vergleichbar zu:

- *UML **Activity Partitions*** (siehe Unterkapitel 3.3.4),
- *BPMN **Swimlanes*** (siehe Unterkapitel 4.3.4) und
- EPK ***Organizational Units*** (siehe Unterkapitel 5.3.4).

Beschreibung:

Ein Akteur (***Participant***) stellt eine Organisationseinheit in einem Geschäftsprozess dar. Dabei handelt es sich um einen Typ oder eine Rolle, die u.a. eine Firma, ein System, eine Personengruppe oder ein einzelnes Individuum bei der Ausführung des Prozesses einnehmen kann (vgl. [Kec06]). Der Verantwortungsbereich eines ***Participants*** wird als ***Participant Partition*** modelliert. Alle in einer ***Participant Partition*** enthaltenen ***Activities*** werden bei der Diagrammausführung durch den zuständigen ***Participant*** ausgeführt.

Unterschiede:

UML ***Activity Partitions***, *BPMN* ***Swimlanes*** und EPK ***Organizational Units*** verfolgen die gleichen Zielsetzungen und unterscheiden sich nur in Implementationsdetails:

1. **Problemstellung**: Die EPK-Notation sieht keine Verschachtelung der Verantwortungsbereiche von Akteuren wie in der *UML*- oder *BPMN*-Notation vor.

   Lösung: Die Verschachtelung von EPK ***Organizational Units*** lässt sich durch einen hierarchischen Namensaufbau der Form *Obertyp/Untertyp/...* nachahmen (siehe Abb. 7.35).

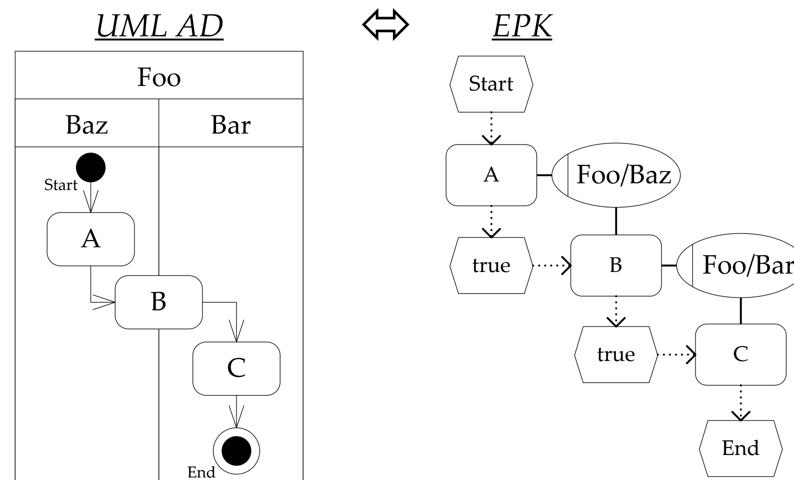


Abbildung 7.35.: Erweiterung der EPK-Notation um EPK ***Organizational Units*** mit hierarchischem Namensaufbau zur Abbildung verschachtelter Verantworungsbereiche.

2. **Problemstellung**: Bei *BPMN* ***Swimlanes*** wird im Gegensatz zu *UML* ***Activity Partitions*** und EPK ***Organizational Units*** weiter zwischen den Untertypen *BPMN* ***Pool*** und *BPMN* ***Lanes*** unterschieden. Es stellt sich die Frage nach der korrekten Abbildung dieser Untertypen in *UML*- und EPK-Notation.

   Lösung: Ein *BPMN* ***Pool*** stellt den Verantwortungsbereiche der obersten (*top-level*) Organisationseinheit dar, d.h. ein *BPMN* ***Pool*** ist in keinem weiteren *BPMN* ***Pool*** enthalten. Ein *BPMN* ***Pool*** entspricht in *UML*-Notation bzw. EPK-Notation der obersten (top-level) *UML* ***Activity Partition*** eines *UML* Aktivitätsdiagrammes bzw. EPK ***Organizational Unit*** einer EPK. *BPMN* ***Lanes*** stellen hingegen die Aufteilung des Zuständigkeitsbereiches eines Akteurs auf mehrere beliebige Untereinheiten dar und sind dementsprechend stets direkt oder indirekt über weitere *BPMN* ***Lanes*** in einem *BPMN* ***Pool*** enthalten. Eine *BPMN* ***Lane*** stellt eine *UML* ***Activity Partition*** bzw. EPK ***Organizational Unit*** dar, welche in einer oder mehreren *UML* ***Activity Partitions*** bzw. EPK ***Organizational Units*** enthalten ist.

3. **Problemstellung**: In einem Geschäftsprozess ist stets zumindest ein Akteur (***Participant***) für die Ausführung der beschriebenen Tätigkeiten zuständig. Die *BPMN*-Notation sieht daher den Einsatz zumindestes eines *BPMN* ***Pools*** in jedem *BPMN* Diagramm vor. Enthält ein *BPMN* Diagramm keinen graphischen, durch den Modellierer festgelegten, *BPMN* ***Pool***,

wird automatisch ein nicht-graphischer *BPMN* ***Pool*** festgelegt, welcher den Namen des *BPMN* Diagrammes übernimmt und alle Notationselemente enthält. Weil im Rahmen dieser Arbeit auf die Benutzung von *BPMN* ***Message Flows*** und einhergehend auf den Einsatz mehrerer *BPMN* ***Pools*** in *BPMN* Diagrammen verzichtet wird (siehe Unterkapitel 4.3.2.1), besitzt ein *BPMN* Diagramm also immer genau einen *BPMN* ***Pool***, also einen top-level Verantwortungsbereich.

In *UML* Aktivitätsdiagrammen bzw. in EPK ist der Einsatz mehrerer top-level *UML* ***Activity Partitions*** bzw. EPK ***Organizational Units*** hingegen möglich, was eine Konvertierung in die *BPMN* erschwert.

Lösung: In Anlehnung an die *BPMN*-Notation müssen *UML* Aktivitätsdiagramme und EPK genau eine top-level *UML* ***Activity Partition*** bzw. EPK ***Organizational Unit*** besitzen.

Wird ein *UML* Aktivitätsdiagramm oder eine EPK mit mehreren top-level Verantwortungsbereichen modelliert, wird bei der Überführung in ein Modell, konform des in dieser Arbeit erstellten *UML*-Metamodells bzw. EPK-Metamodells, automatisch eine weitere *UML* ***Activity Partition*** bzw. EPK ***Organizational Unit*** erstellt, welche alle weiteren *UML* ***Activity Partitions*** bzw. EPK ***Organizational Units*** enthält.

Enthält ein *UML* Aktivitätsdiagramm oder eine EPK dagegen nur einen top-level Verantwortungsbereich, dann wird dieser bei der Überführung in ein Modell, konform des in dieser Arbeit erstellten *UML*-Metamodells bzw. EPK-Metamodells, als oberster Verantwortungsbereich erhalten.

In beiden Fällen, ist somit sichergestellt, dass bei der Konvertierung in ein *BPMN* Diagramm stets immer genau ein *BPMN* ***Pool*** als Abbild des top-level Verantwortungsbereichs existiert.

Ein Referenzmodell, konform des in dieser Arbeit erstellten Referenzmetamodells, enthält genau eine top-level ***Participant Partition***.

7.5. Annotations

Vergleichbar zu:

- *UML* ***Comments*** (siehe Unterkapitel 3.3.5),
- *BPMN* ***Text Annotations*** (siehe Unterkapitel 4.3.5) und
- EPK ***Descriptions*** (siehe Unterkapitel 5.3.5).

Beschreibung:

Annotations werden eingesetzt, um weitere Informationen über Notationselemente, den Ausführungsfluss, oder Implementierungsdetails anzugeben. Es lässt sich aber prinzipiell jede beliebige Anmerkung (***Annotation***) ausdrücken. ***Annotation*** können nahezu jedes andere Notationselement annotieren oder frei stehen, dann annotieren sie implizit den ***Sub-Process*** in dem sie sich befinden.

Unterschiede:

UML ***Comments***, *BPMN* ***Text Annotations*** und EPK ***Descriptions*** erfüllen die gleiche Aufgabe. Außer durch ihre graphische Darstellung unterscheiden sie sich bei den Elementtypen die sie annotieren können:

1. **Problemstellung**: Während *UML* ***Comments*** und EPK ***Descriptions*** jedes Notationselement der jeweiligen Sprache annotieren können, dürfen *BPMN* ***Text Annotations*** nur *BPMN* ***Activities*** und *BPMN* ***Flow Controls*** annotieren.

   Lösung: *UML* ***Comments*** dürfen nur *UML* ***Activity Nodes*** und *UML* ***Flow Controls***, EPK ***Descriptions*** nur EPK ***Functions*** sowie EPK ***Flow Controls*** und ***Annotations*** nur ***Activities*** und ***Flow Controls*** annotieren.

7.6. Vergleichstabelle

Die nachstehende Tabelle führt alle Notationselemente für die Geschäftsprozessmodellierung in *SOAMIG* auf, die im Rahmen dieser Arbeit aus den Sprachspezifikationen übernommen oder als Ergänzung der Spezifikationen eingeführt wurden.

Referenz	***UML AD***	***BPMN***	**EPK**
Activities	Activity Nodes	Activities	Functions
Atomic Process	Action	Task	Atomic Function
Sub-Process	Activity	Sub-Process	Complex Function
Call Process	Call Action	Call Task	Call Function
Flow Controls	Flow Controls	Flow Controls	Flow Controls
Start Event	Initial Node	Start Event	Start Event
Flow Final Event	Flow Final Node	End Event	End Event
Process Final Event	Activity Final Node	Terminate End Event	Terminate End Event
Signal Throwing Event	Send Signal Action	Signal Throwing Event	Signal Throwing Event
Signal Caching Event	Accept Signal Action	Signal Catching Event	Signal Catching Event
Resource Throwing Event	Send Resource Action	Resource Throwing Event	Resource Throwing Event
Resource Catching Event	Accept Resource Action	Resource Catching Event	Resource Catching Event
Flow	Control Flow	Sequence Flow	Control Flow
Resource Flow	Object Flow	Sequence Flow mit Data Objects	Control Flow mit Information Objects
Exclusive Branch Connector	Decision Node	Data-based Exclusive Branch Gateway	Exclusive Branch Connector
Parallel Branch Connector	Fork Node	Parallel Branch Gateway	Parallel Branch Connector
Exclusive Merge Connector	Merge Node	Data-based Exclusive Merge Gateway	Exclusive Merge Connector
Parallel Merge Connector	Join Node	Parallel Merge Gateway	Parallel Merge Connector
Resources	Objects	Data Objects	Information Objects
Participant Partitions	Activity Partitions	Swimlanes	Organizational Units
top-level Participant Partition	top-level Activity Partition	Pool	top-level Organizational Unit
verschachtelte Participant Partition	*verschachtelte* Activity Partition	Lane	*verschachtelte* Organizational Unit
Annotations	Comments	Text Annotations	Descriptions

Tabelle 7.1.: Vergleichstabelle der Notationselemente in den jeweiligen Modellierungssprachen und der Referenz.

8. Erhebung und Beschreibung von Metamodellen

Auf Basis der in den letzten Kapiteln vorgestellten (siehe Kapitel 3.3, 4.3 und 5.3) und in Anlehnung an die aneinander angepassten Sprachkonzepte und Notationselemente (siehe Unterkapitel 7) der *UML* Aktivitätsdiagramme, der *BPMN*-, EPK- und Referenz-Notation, werden im folgenden Metamodelle für diese Notationen beschrieben. Die Metamodelle und die damit abgebildeteten Sprachkonzepte sind für die Geschäftsprozessmodellierung im Rahmen des *SOAMIG*-Projekts ausgelegt. Zum entsprechenden Metamodell konforme *UML*-, *BPMN* und EPK-Geschäftsprozessmodelle lassen sich anhand passender Konverter (siehe Teil IV) in semantisch-äquivalente Modelle der jeweils anderen Notationen oder in ein semantisch-äquivalentes, referenzmetamodellkonformes Referenzmodell transformieren.

Aufgrund der Auswahl sich stark ähnelnder Sprachkonzepte und Notationselemente aus den jeweiligen Geschäftsprozessmodellierungssprachen und der Abbildung in Referenzkonzepte und -elemente besitzen die Diagramme der verschiedenen Metamodelle einen nahezu identischen Aufbau und die Klassen ähnliche Attribute und Assoziationen.

8.1. Übersicht

Als oberste Klassen aller Metamodelle stehen die ***Element-*** und ***FlowElement-***Klassen (siehe Abb. 8.1, 8.2, 8.3 und 8.4).

Die ***Element***-Klasse stellt Attribute und Assoziationen für die eindeutige Identifizierung von Elementinstanzen in Modellen, die Namensgebung dieser Instanzen, sowie optionale textuelle Anmerkungen dieser Instanzen anhand von Instanzen der *UML* ***Comment-***, *BPMN* ***TextAnnotation-***, EPK ***Description***- bzw. ***Annotation-***Klasse bereit.

Die ***FlowElement-***Klasse fasst die Sprachelementklassen zusammen, deren Instanzen über UML ***ControlFlow-***/ *UML* ***ObjectFlow-***, *BPMN* ***SequenceFlow-***, EPK ***ControlFlow***- bzw. ***Flow-*** / ***ResourceFlow***- Klasseninstanzen in den Ausführungsfluss eingebunden werden. ***FlowElement***-Instanzen stellen Knoten in Geschäftsprozessmodellen, in Anlehnung an den Knotenbegriff aus der Graphentheorie, dar. Diese Zusammenfassung dient lediglich dazu, die visuelle Notation der Metamodelle zu entlasten und die Übersichtlichkeit der Diagramme zu erhöhen.

Die Spezialisierungen der **`Element`**- und **`FlowElement`**-Klassen werden in den weiteren Unterkapiteln näher beschrieben und durch differenziertere Diagramme dargestellt. Die Unterkapitelstruktur orientiert sich an den in Geschäftsprozessen identifizierten Hauptmerkmalen (siehe Kapitel II) und der bereits in vorherigen Kapiteln angewendeten Struktur. Die Aufteilung erfolgt in die folgenden Bereiche:

- ***Activities*** (siehe Unterkapitel 8.2),
- ***Flow Controls*** (siehe Unterkapitel 8.3),
- ***Resources*** (siehe Unterkapitel 8.4) und
- ***Participant Partitions*** (siehe Unterkapitel 8.5).

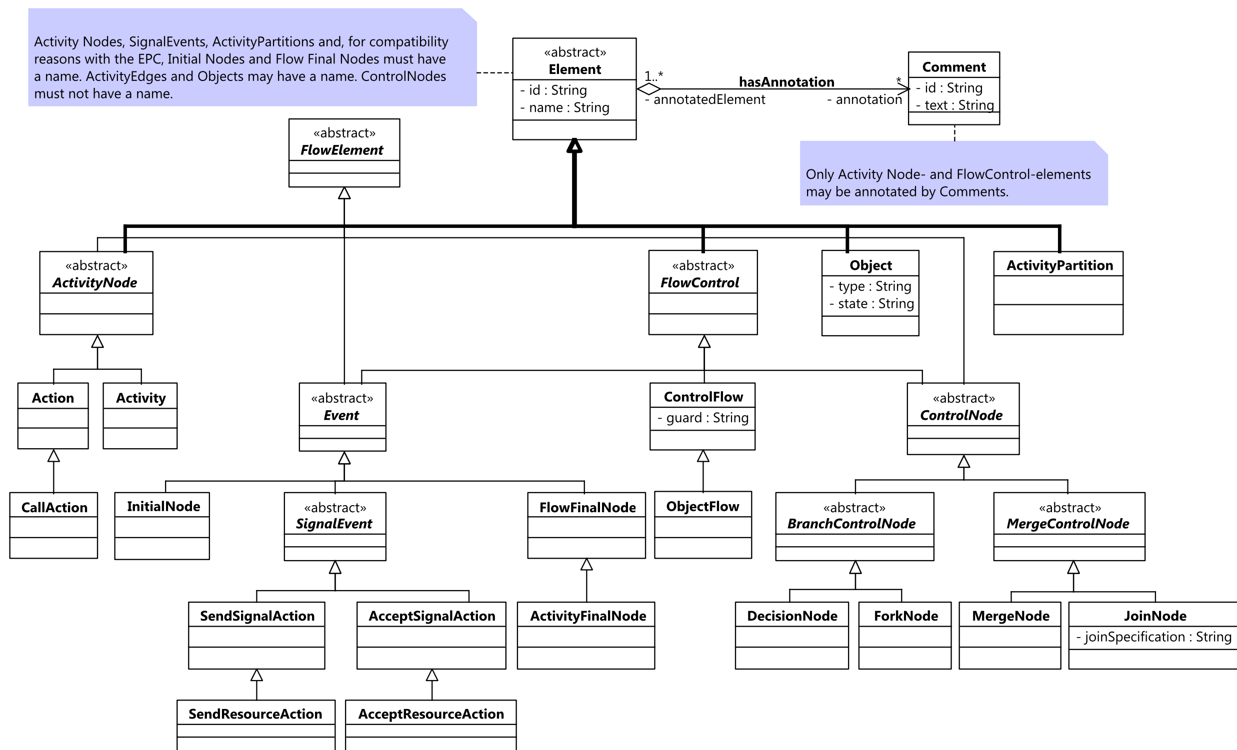


Abbildung 8.1.: Übersicht der Notationselemente im *UML*-Metamodell.

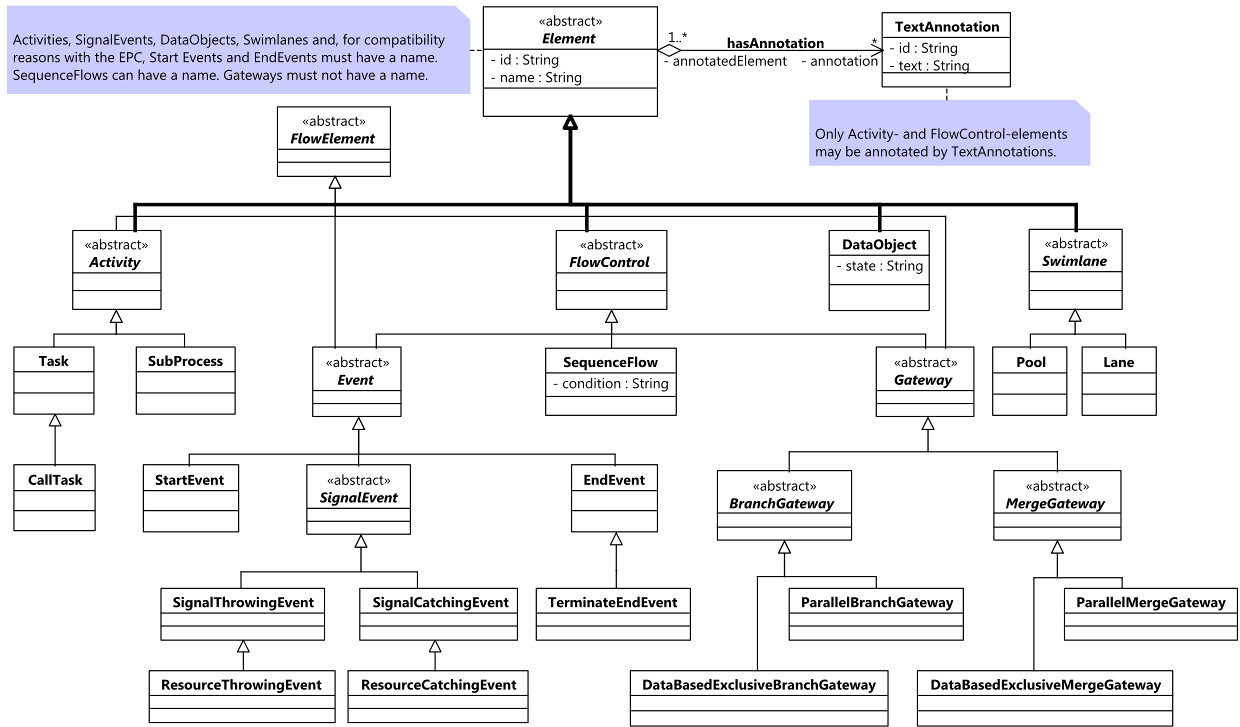


Abbildung 8.2.: Übersicht der Notationselemente im *BPMN*-Metamodell.

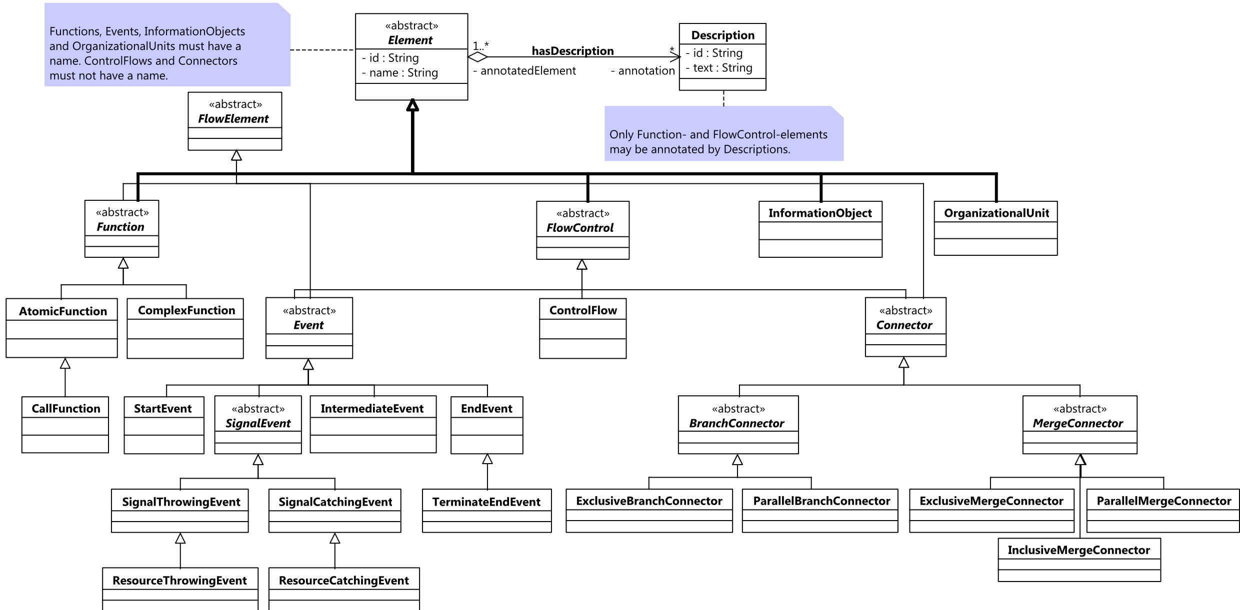


Abbildung 8.3.: Übersicht der Notationselemente im EPK-Metamodell.

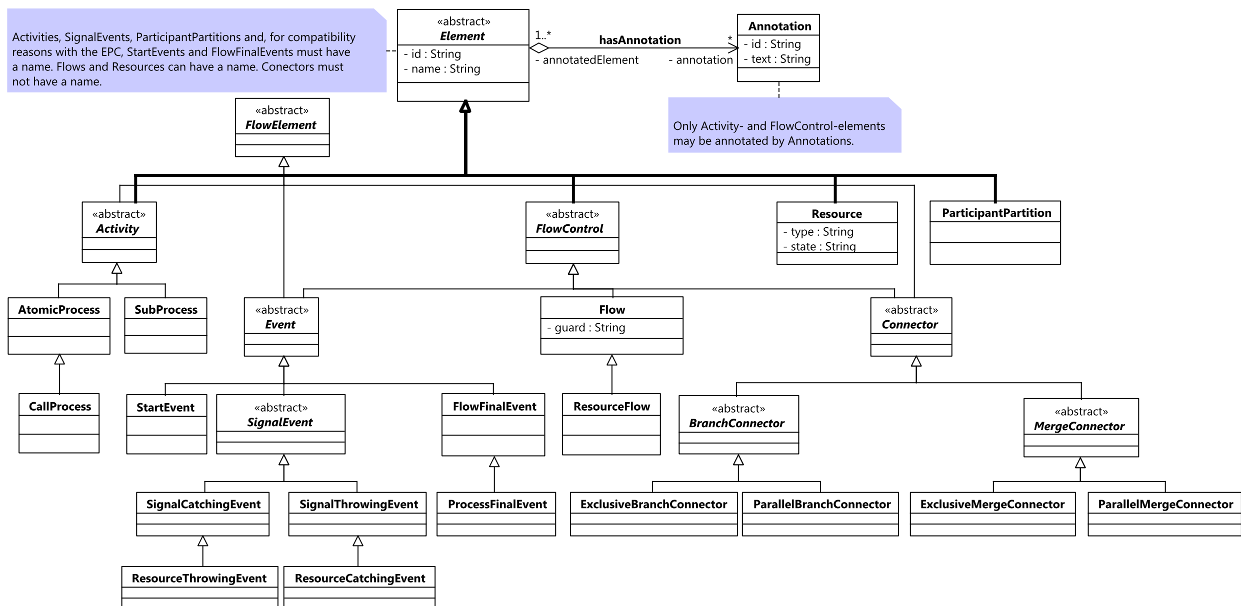


Abbildung 8.4.: Übersicht der Notationselemente im Referenzmetamodell.

8.2. *Activities*

In der *UML* **ActivityNode-**, *BPMN* **Activity-**, EPK **Function**- bzw. **Activity**-Klasse zusammengefasst befinden sich die unterschiedlichen Ausprägungen aktiver Elemente – d.h. durch Akteure ausführbare Tätigkeiten – eines Geschäftsprozessesmodells (siehe Abb. 8.5, 8.6, 8.7 und 8.8).

UML **Action**-, *BPMN* **Task**-, EPK **AtomicFunction**- und **AtomicProcess**-Klasseninstanzen entsprechen atomaren Tätigkeiten in Geschäftsprozessmodellen. *UML* **Action**-, *BPMN* **Task**- bzw. **AtomicProcess**-Instanzen werden über mindestens eine ein- und mindestens eine ausgehende *UML* **ControlFlow**-/ *UML* **ObjectFlow**-, *BPMN* **SequenceFlow**- bzw. **Flow**-/ **ResourceFlow**-Instanz in den Ausführungsfluss eingebunden. EPK **Function**-Instanzen werden hingegen über genau eine ein- und eine ausgehende EPK **ControlFlow**-Instanz in den Ausführungsfluss einer EPK eingebunden.

Instanzen der *UML* **Activity**-, *BPMN* **SubProcess**-, EPK **ComplexFunction**- und **SubProcess**-Klasse entsprechen hingegen nicht atomaren, aus weiteren Notationselementen zusammengesetzten Tätigkeiten. *UML* **Activity**-, *BPMN* **SubProcess**- bzw. **SubProcess**-Instanzen werden über mindestens eine ein- und mindestens eine ausgehende *UML* **ControlFlow**-/ *UML* **ObjectFlow**-, *BPMN* **SequenceFlow**- bzw. **Flow**-/ **ResourceFlow**-Instanz in den Ausführungsfluss eingebunden. EPK **ComplexFunction**-Instanzen werden hingegen über genau eine ein- und eine ausgehende EPK **ControlFlow**-Instanz in den Ausführungsfluss einer EPK eingebunden. Beschreibt eine *UML* **Activity**-, *BPMN* **SubProcess**-, EPK **ComplexFunction**- oder **SubProcess**-Instanz ein gesamtes Geschäftsprozessmodell, besitzt dieses Element keine

ein- und keine ausgehenden Ausführungsflussinstanzen. *UML* **`Activity`**-, *BPMN* **`SubProcess`**-, EPK **`ComplexFunction`**- bzw. **`SubProcess`**-Instanzen besitzen genau einen obersten/ allgemeinen (*top-level*) Verantwortungsbereich/ eine *top-level UML* **`ActivityPartition`**-, *BPMN* **`Pool`**-, EPK **`OrganizationalUnit`**- bzw. **`ParticipantPartition`**-Instanz, dessen/ deren dafür zuständiger Akteur alle darin enthaltenen Tätigkeiten ausführt (siehe Unterkapitel 7.4).

Eine *UML* **`CallAction`**-, *BPMN* **`CallTask`**-, EPK **`CallFunction`**- oder **`CallProcess`**-Instanz ruft bei ihrer Ausführung das Verhalten eines *UML* Aktivitätsdiagramms, *BPMN* Diagramms, einer EPK oder eines Referenzdiagramms auf.

Abbildung 8.5.: Diagramm der *UML* **`ActivityNode`**-Klassen mit Assoziationen.

Abbildung 8.6.: Diagramm der *BPMN* **`Activity`**-Klassen mit Assoziationen.

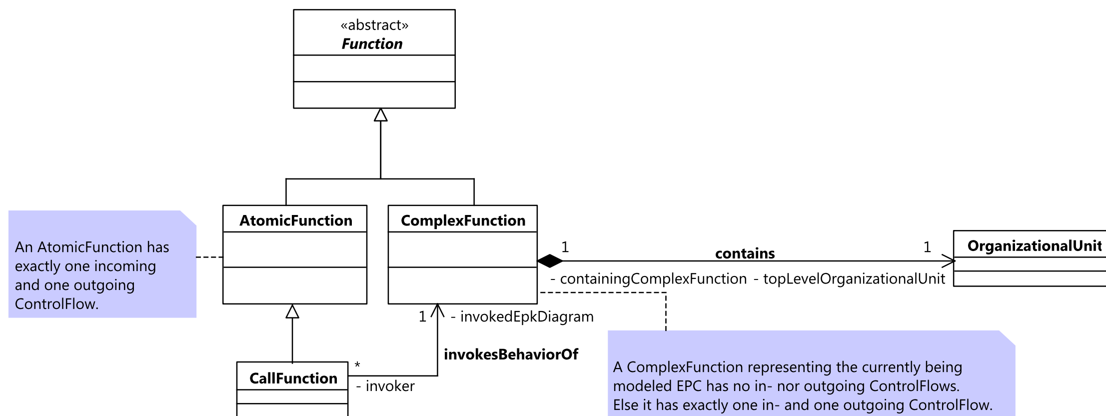


Abbildung 8.7.: Diagramm der EPK ***Function***-Klassen mit Assoziationen.

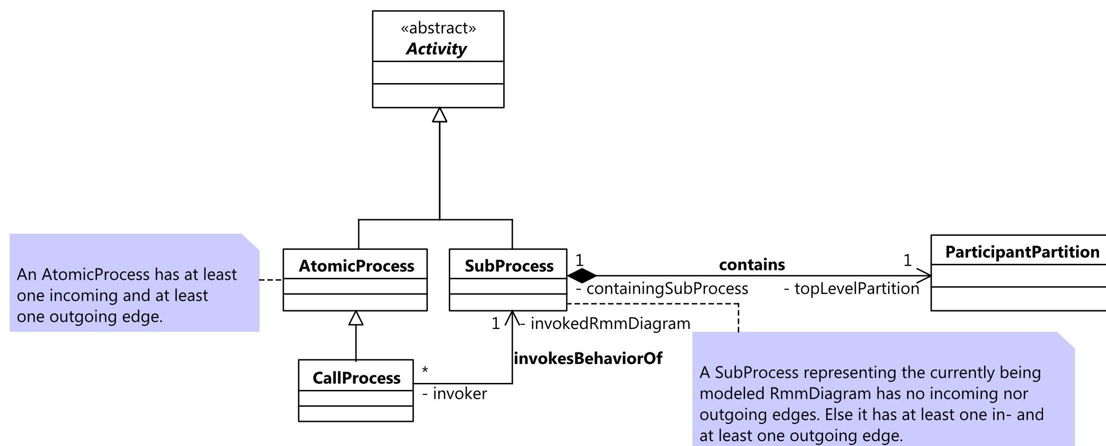


Abbildung 8.8.: Diagramm der ***Activity***-Klassen mit Assoziationen im Referenzmetamodell.

8.3. *Flow Controls*

Die ***FlowControl***-Klassen und deren Spezialisierungen sind für die Steuerung des Ausführungsflusses zuständig. Es wird unterschieden zwischen:

- ***Event***-Klassen (siehe Unterkapitel 8.3.1),
- ***Flow***-Klassen (siehe Unterkapitel 8.3.2) und
- ***Connector***-Klassen (siehe Unterkapitel 8.3.3).

8.3.1. *Events*

Instanzen der ***Event***-Klassen stellen das Auftreten von außerhalb oder innerhalb des Diagrammes/ Geschäftsprozesses eingeläuteter Hergänge dar, welche sich auf den Ausführungfluss auswirken (siehe Abb. 8.9, 8.10, 8.11 und 8.12).

Für den Start von Ausführungsflüssen in Geschäftsprozessmodellen sind primär *UML* ***InitialNode***-, *BPMN* ***StartEvent***-, EPK ***StartEvent***- bzw. ***StartEvent***-Instanzen zuständig. *UML* ***InitialNode***-, *BPMN* ***StartEvent***- und ***StartEvent***-Instanzen werden über mindestens eine ausgehende *UML* ***ControlFlow***-, *BPMN* ***SequenceFlow***- bzw. ***Flow***-Instanz in den Ausführungsfluss eingebunden. EPK ***StartEvent***-Instanzen werden hingegen über genau eine ausgehende EPK ***ControlFlow***-Instanz in den Ausführungsfluss einer EPK eingebunden. Aus Kompatibilitätsgründen zur *UML* und in Anlehnung an die Lösung der 2. Problemstellung in Unterkapitel 7.2.1.1, dürfen *UML* ***InitialNode***-, *BPMN* ***StartEvent***-, EPK ***StartEvent***- bzw. ***StartEvent***-Instanzen nicht der Ursprung einer *UML* ***ObjectFlow***-Instanz, einer *BPMN* ***SequenceFlow***-Instanz, welche *BPMN* ***DataObject***-Instanzen transportiert, einer EPK ***ControlFlow***-Instanz welche indirekt EPK ***InformationObject***-Instanzen transportiert , bzw. einer ***ResourceFlow***-Instanz, sein.

Alternativ können die Ausführungsflüsse von Geschäftsprozessen auch über *UML* ***AcceptSignalAction***-, *BPMN* ***SignalCatchingEvent***-, EPK ***SignalCatchingEvent***, ***SignalCatchingEvent***-Instanzen gestartet werden. Instanzen dieser Klassen reagieren auf das Eintreffen eines asynchronen Signals, welches von einer oder mehreren namentlich zugehörigen Instanzen der *UML* ***SendSignalAction***-, *BPMN* ***SignalThrowingEvent***-, EPK ***SignalThrowingEvent***-, ***SignalThrowingEvent***-Klasse ausgelöst wird. *UML* ***AcceptSignalAction***-, *BPMN* ***SignalCatchingEvent***- und ***SignalCatchingEvent***-Instanzen werden über beliebig viele ein- und mindestens eine ausgehende *UML* ***ControlFlow***-/ *UML* ***ObjectFlow***-, *BPMN* ***SequenceFlow***- bzw. ***Flow***-/ ***ResourceFlow***-Instanz in den Ausführungsfluss eines Diagrammes eingebunden. EPK ***SignalCatchingEvent***-Instanzen werden hingegen über maximal eine ein- und genau eine ausgehende EPK ***ControlFlow***-Instanz in den Ausführungsfluss einer EPK eingebunden.

Die Gegenstücke bilden Instanzen der *UML* ***SendSignalAction***-, *BPMN* ***SignalThrowingEvent***-, EPK ***SignalThrowingEvent***-, ***SignalThrowingEvent***-Klasse, welche bei ihrer Ausführung ein asynchrones Signal aussenden. *UML* ***SendSignalAction***-, *BPMN* ***SignalThrowingEvent***- und ***SignalThrowingEvent***-Instanzen werden über mindestens eine ein- und beliebig viele ausgehende *UML* ***ControlFlow***-/ *UML* ***ObjectFlow***-, *BPMN* ***SequenceFlow***- bzw. ***Flow***-/ ***ResourceFlow***-Instanz in den Ausführungsfluss eines Diagrammes eingebunden. EPK ***SignalThrowingEvent***-Instanzen werden hingegen über genau eine ein- und maximal eine ausgehende EPK ***ControlFlow***-Instanz in den Ausführungsfluss einer EPK eingebunden. Steht eine Instanz dieser Klassen am Ende eines Ausführungsflusses enden alle eingehenden Ausführungsflusse an ihr.

Instanzen der *UML* ***SendResourceAction***-, *BPMN* ***ResourceThrowingEvent***-, EPK ***ResourceThrowingEvent***- bzw. ***ResourceThrowingEvent***-Klasse ermöglichen hingegen den Austausch von Daten zwischen Notationselementen in verschiedenen *UML* ***Activity***-, *BPMN* ***SubProcess***-, EPK ***ComplexFunction***- bzw. ***SubProcess***-Instanzen (siehe Lösung der 1. Problemstellung in Unterkapitel 7.3). Instanzen dieser Klassen stehen stets am Ende des Ausführungsflusses und beenden alle eingehenden Ausführungsflüsse.

Als passende Gegenstücke zu Instanzen der *UML* ***SendResourceAction***-, *BPMN* ***ResourceThrowingEvent***-, EPK ***ResourceThrowingEvent***- und ***ResourceThrowingEvent***-Klassen nehmen die namentlich zugehörigen Instanzen der *UML* ***AcceptResourceAction***-, *BPMN* ***ResourceCatchingEvent***-, EPK ***ResourceCatchingEvent*** bzw. ***ResourceCatchingEvent***-Klasse die übertragenen Datensätze entgegen. Instanzen dieser Klasse können alternativ als Startpunkte von Geschäftsprozessmodellen dienen.

Zur Beendigung der eingehenden Ausführungsflüsse dienen Instanzen der *UML* ***FlowFinalNode***-, *BPMN* ***EndEvent***-, EPK ***EndEvent***-, ***FlowFinalEvent***-Klasse. *UML* ***FlowFinalNode***-, *BPMN* ***EndEvent***- und ***FlowFinalEvent***-Instanzen werden über mindestens eine eingehende *UML* ***ControlFlow***-, *BPMN* ***SequenceFlow***- bzw. ***Flow***-Instanz in den Ausführungsfluss des Diagrammes eingebunden. EPK ***EndEvent***-Instanzen werden hingegen über genau eine eingehende EPK ***ControlFlow***-Instanz in den Ausführungsfluss einer EPK eingebunden. Aus Kompatibilitätsgründen zur *UML* und in Anlehnung an die Lösung der 2. Problemstellung in Unterkapitel 7.2.1.2, dürfen *UML* ***FlowFinalNode***-, *BPMN* ***EndEvent***-, EPK ***EndEvent***- bzw. ***EndEvent***-Instanzen nicht das Ziel einer *UML* ***ObjectFlow***-Instanz, einer *BPMN* ***SequenceFlow***-Instanz, welche *BPMN* ***DataObject***-Instanzen transportiert, einer EPK ***ControlFlow***-Instanz, welche EPK ***InformationObject***-Instanzen transportiert , bzw. einer ***ResourceFlow***-Instanz sein.

Die gleichen Einschränkungen gelten für Instanzen der *UML* ***ActivityFinalNode***-, *BPMN* ***TerminateEndEvent***-, EPK ***TerminateEndEvent***-, ***ProcessFinalEvent***-Klasse, welche die Ausführung des gesamten Diagrammes stoppen wenn ein Ausführungsfluss in sie einläuft (siehe Lösung der 3. Problemstellung in Unterkapitel 7.2.1.3).

Im Vergleich zum *UML*-, *BPMN*-Metamodell und Referenzmetamodell bietet das EPK-Metamodell als weitere ***Event***-Spezialisierung die ***IntermediateEvent***-Klasse. Instanzen dieser Klasse werden verwendet um die Ausführungsbedingungen von EPK ***ControFlow***-Instanzen zu speichern. Sie werden über genau eine ein- und genau eine ausgehenden EPK ***ControlFlow***-Instanz in den Ausführungsfluss einer EPK gebunden und folgen im Ausführungsfluss direkt oder indirekt, über EPK ***Connector***-Instanzen, Instanzen der EPK ***Function***-Unterklassen.

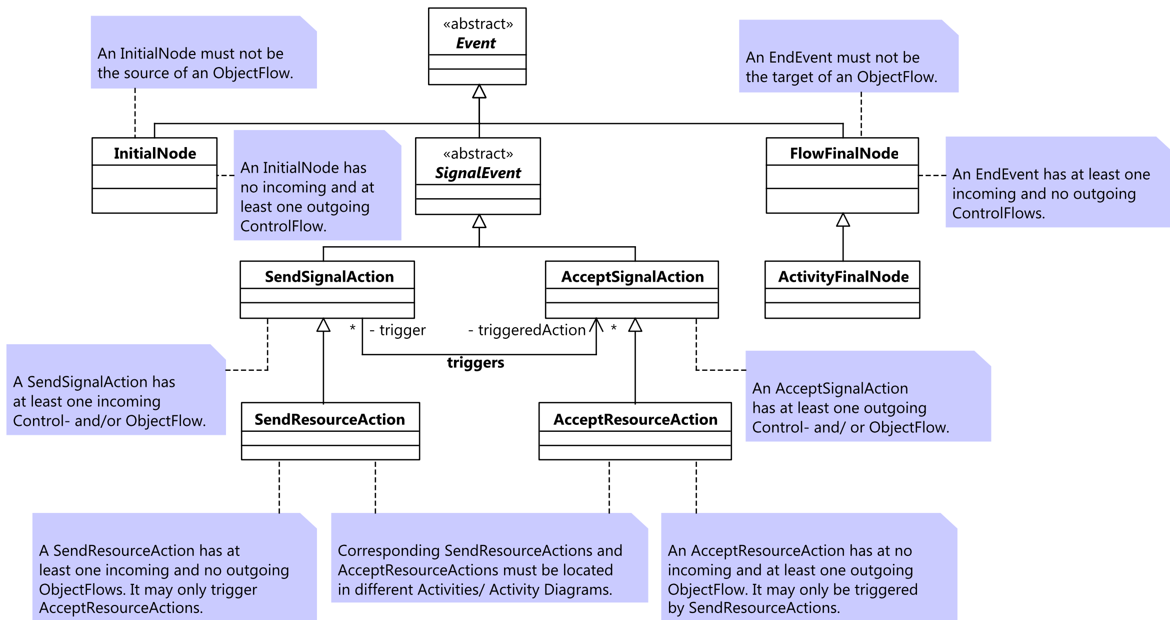


Abbildung 8.9.: Diagramm der *UML* **`Event`**-Klassen mit Assoziationen.

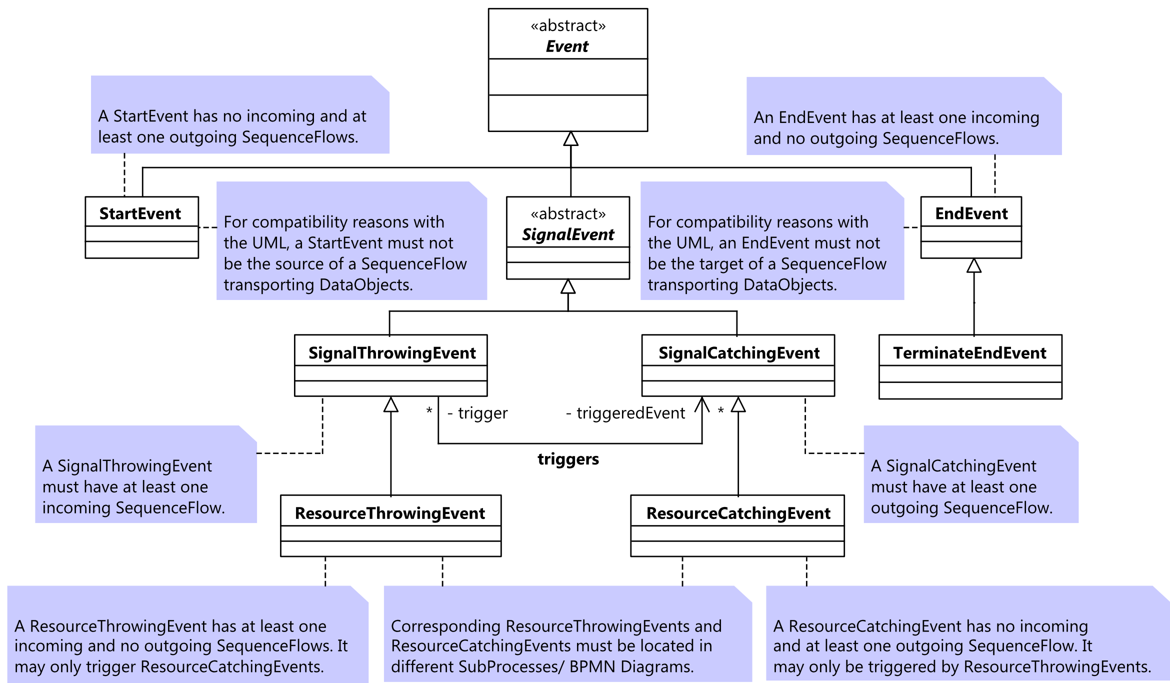


Abbildung 8.10.: Diagramm der *BPMN* **`Event`**-Klassen mit Assoziationen.

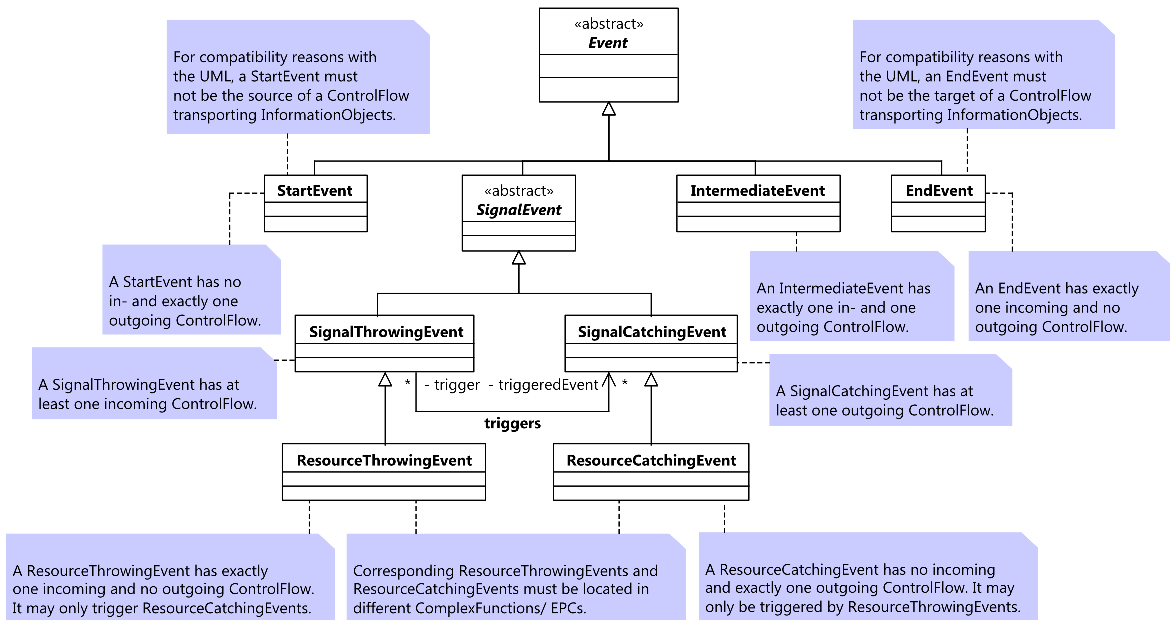


Abbildung 8.11.: Diagramm der EPK **`Event`**-Klassen mit Assoziationen.

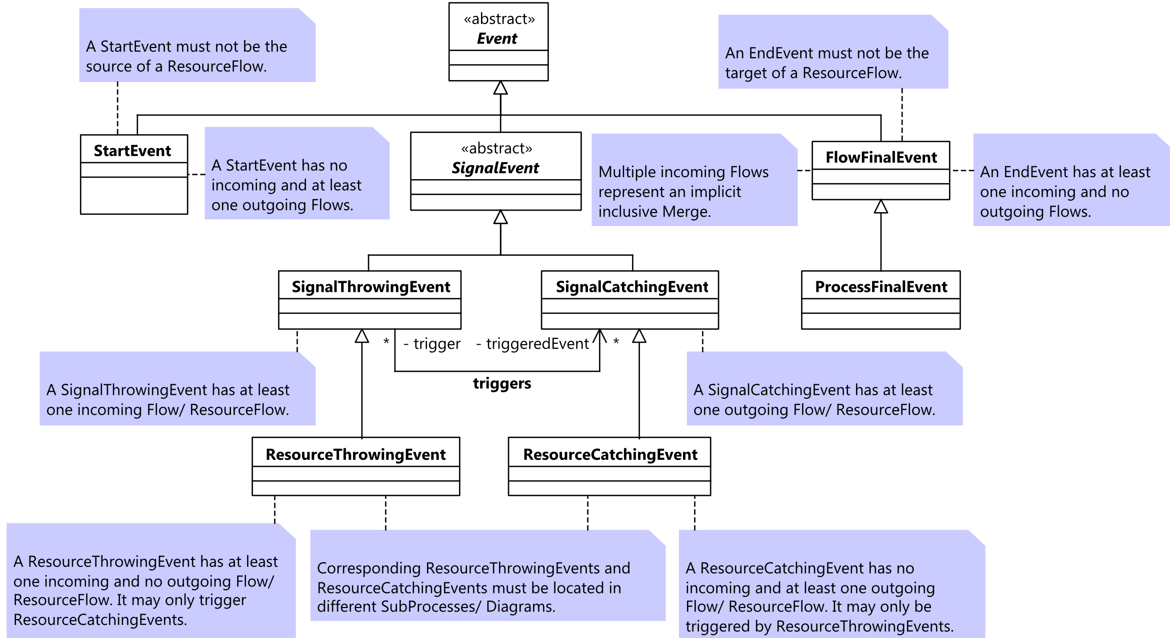


Abbildung 8.12.: Diagramm der **`Event`**-Klassen mit Assoziationen im Referenzmetamodell.

8.3.2. *Flows*

Der Ausführungsfluss von Diagrammen wird anhand von Instanzen der *UML* ***ControlFlow-***, *UML* ***ObjectFlow-***, *BPMN* ***SequenceFlow-***, EPK ***ControlFlow-***, ***Flow-*** und ***ResourceFlow-***Klassen gesteuert (siehe Abb. 8.13, 8.14, 8.15 und 8.16). Es handelt sich um gerichtetete Verbindungen zwischen einem Ursprungs- und einem Endknoten im gleichen Unterprozess, beide Verbindungspunkte sind *UML* ***FlowElement-***, *BPMN* ***FlowElement-***, EPK ***FlowElement-*** oder ***FlowElement***-Insanzen. Anzumerken ist, dass sich durch die bipartite Aufteilung von EPK, EPK ***ControlFlow***-Instanzen abwechselnd EPK ***Function-*** und EPK ***Event***-Instanzen, möglicherweise über EPK ***Connector***-Instanzen hinweg, verbinden.

Instanzen der *UML* ***ControlFlow-***, *UML* ***ObjectFlow-***, *BPMN* ***SequenceFlow-***, ***Flow-*** und ***ResourceFlow***-Klassen besitzen ein optionales Attribut für einen Bezeichner, welchen sie von ihrer jeweiligen ***Element***-Oberklasse erben und ein Attribut für eine optionale Ausführungsbedingung. Der Bezeichner und die Ausführungsbedingung von EPK ***ControlFlow***-Instanzen werden hingegen nicht als Attribute von EPK ***ControlFlow***-Instanzen sondern als Attribute von EPK ***IntermediateEvent***-Instanzen gespeichert (siehe 5.3.2.1).

Die *UML* ***ObjectFlow-*** bzw. ***Resource***-Klassen stellen Spezialisierungen der *UML* ***ControlFlow-*** bzw. ***Flow***-Klassen dar, deren Instanzen zusätzlich zum Ausführungsfluss für den Austausch von Daten zwischen Tätigkeiten im Geschäftsprozess zuständig sind. Im *BPMN*- und EPK-Metamodell existieren keine äquivalente Spezialisierungen, stattdessen übernehmen *BPMN* ***SequenceFlow-*** bzw. EPK ***ControlFlow***-Instanzen den optionalen Transport von Daten (siehe Lösung zur 1. Problemstellung in Unterkapitel 7.2.2.1).

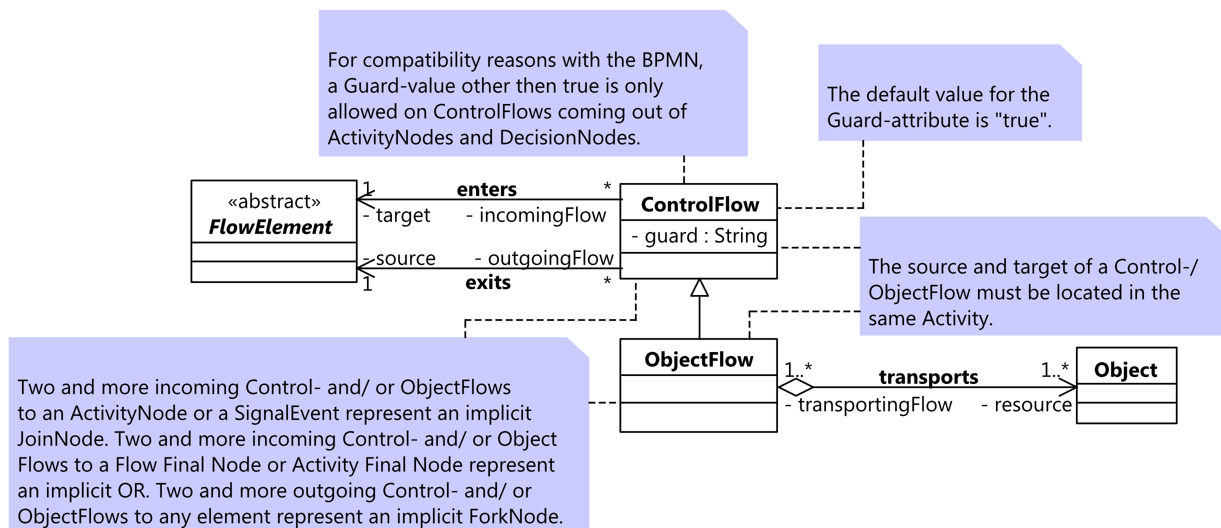


Abbildung 8.13.: Diagramm der *UML* ***ControlFlow*** und ***ObjectFlow***-Klassen mit Attributen und Assoziationen.

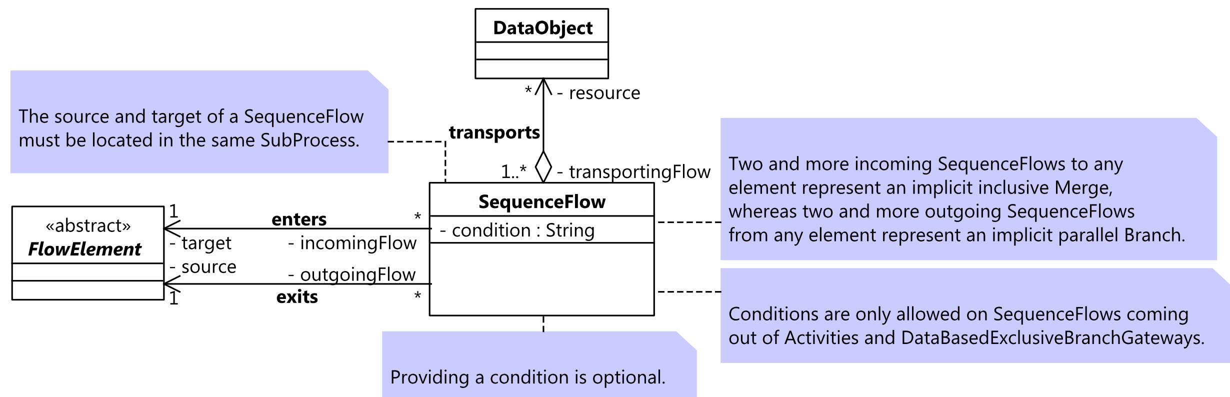


Abbildung 8.14.: Diagramm der *BPMN* **`SequenceFlow`**-Klasse mit Attributen und Assoziationen.

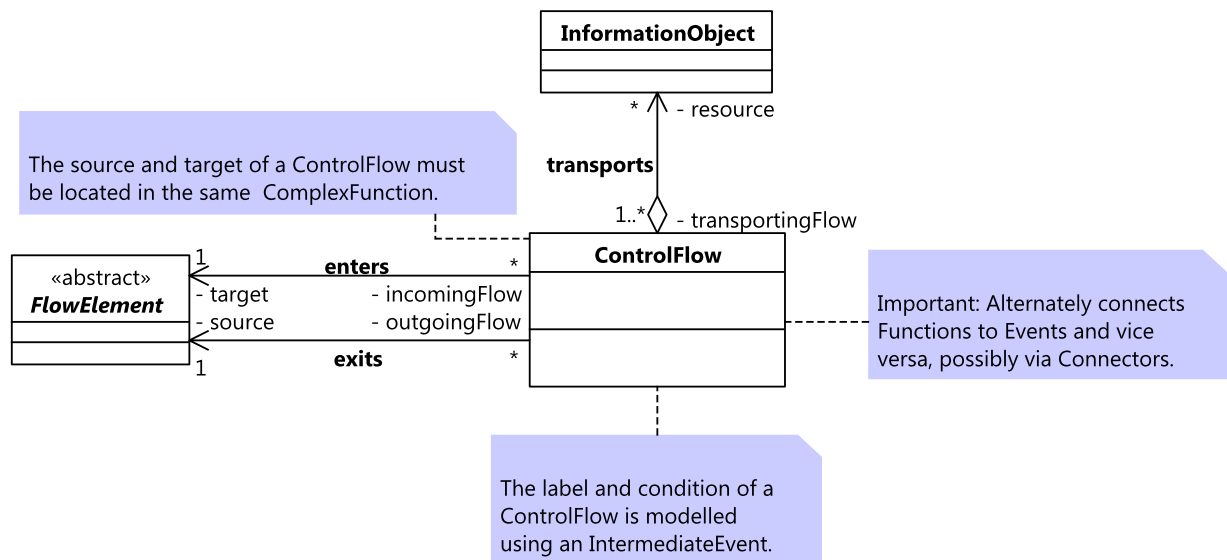


Abbildung 8.15.: Diagramm der EPK **`ControlFlow`**-Klasse mit Assoziationen.

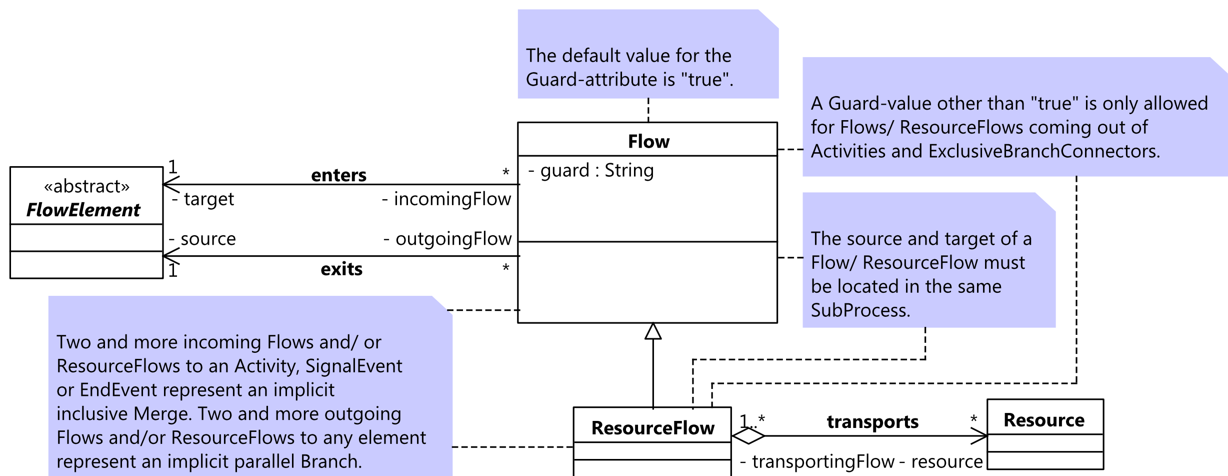


Abbildung 8.16.: Diagramm der ***Flow***-Klassen mit Attributen und Assoziationen im Referenzmetamodell.

8.3.3. *Connectors*

Das Auseinanderfließen und der Zusammenfluss mehrerer Ausführungsflüsse wird über Instanzen *UML ControlNode*-, *BPMN Gateway*-, EPK *Connector*-, *Connector*-Spezialisierungen geregelt, den *UML BranchControlNode*-, *BPMN BranchGateway*-, EPK *BranchConnector*- bzw. *BranchConnector*- und *UML* ***MergeControlNode***-, *BPMN* ***MergeGateway***-, *EPK* ***MergeConnector***- bzw. ***Merge-Connector***-Klassen (siehe Abb. 8.17, 8.18, 8.19 und 8.20). Instanzen beider Spezialisierungen unterscheiden sich durch die Anzahl ein- und ausgehender Ausführungsflüsse und lassen sich nach der Art, wie sie Ausführungsflüsse aufteilen bzw. zusammenführen, weiter unterteilen. Im *UML*-, *BPMN*-Metamodell und Referenzmetamodell wird ausschließlich zwischen exklusiven und parallelen *UML* ***ControlNode***-, *BPMN* ***Gateway***- bzw. ***Connector***-Instanzen unterschieden. Die EPK bietet aus Kompatibilitätsgründen zusätzlich zu exklusiven und parallen noch inklusive EPK-***Connector***-Instanzen (siehe Punkt g der Lösung zur 1. Problemstellung in Unterkapitel 7.2.2). In Anlehnung an die Lösung dieses Problems, bieten Instanzen der *UML* ***JoinNode***-Klasse das ***JoinSpecification***-Attribut, das sich mit einem inklusivem Ausdruck für den Zusammenfluss der Ausführungsflüsse belegen lässt (siehe Punkt e der Lösung zur 1. Problemstellung in Unterkapitel 7.2.2).

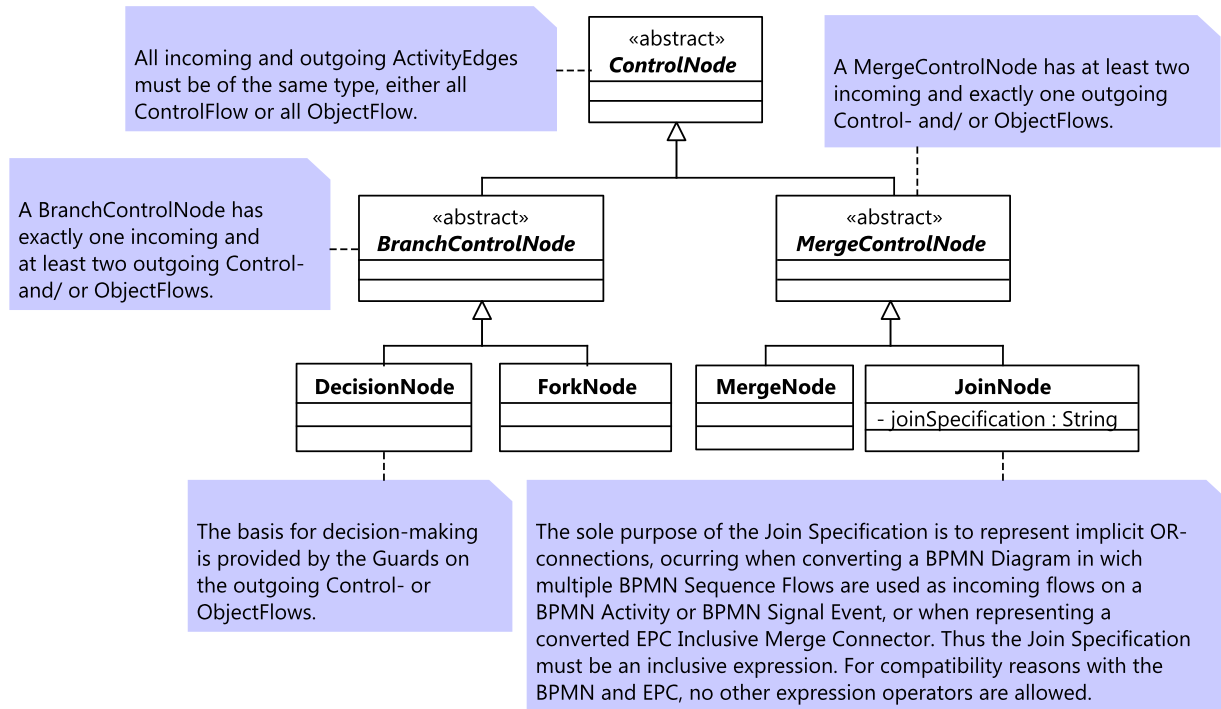


Abbildung 8.17.: Diagramm der *UML* **`ControlNode`**-Klassen mit Attributen.

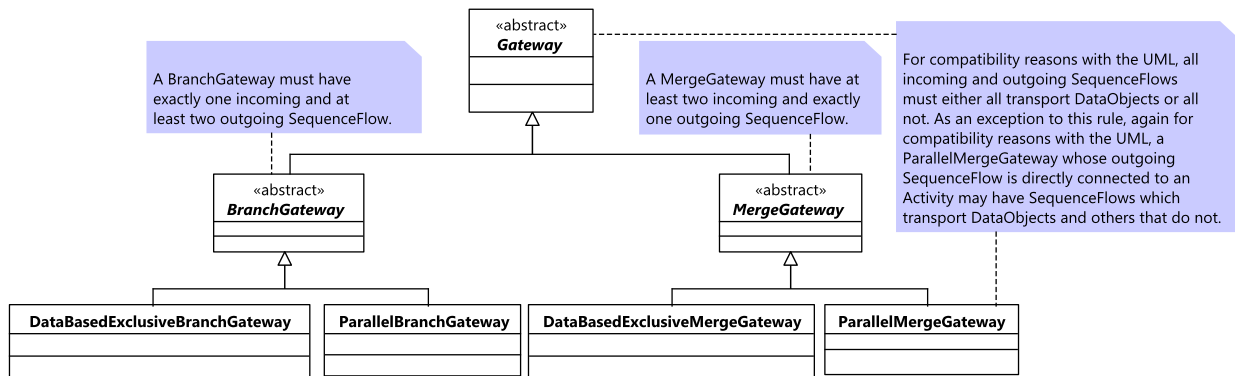


Abbildung 8.18.: Diagramm der *BPMN* **`Gateway`**-Klassen.

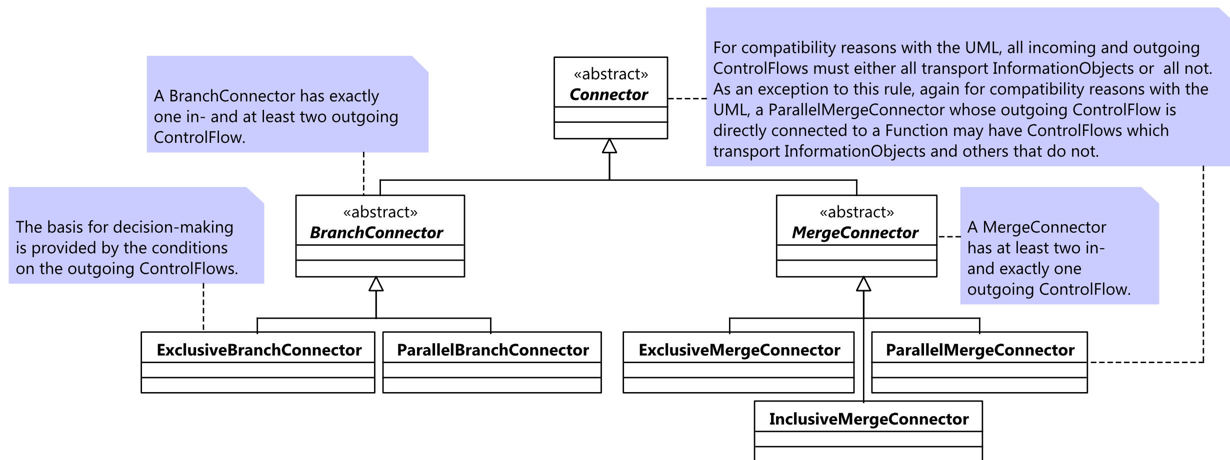


Abbildung 8.19.: Diagramm der EPK **`Connector`**-Klassen.

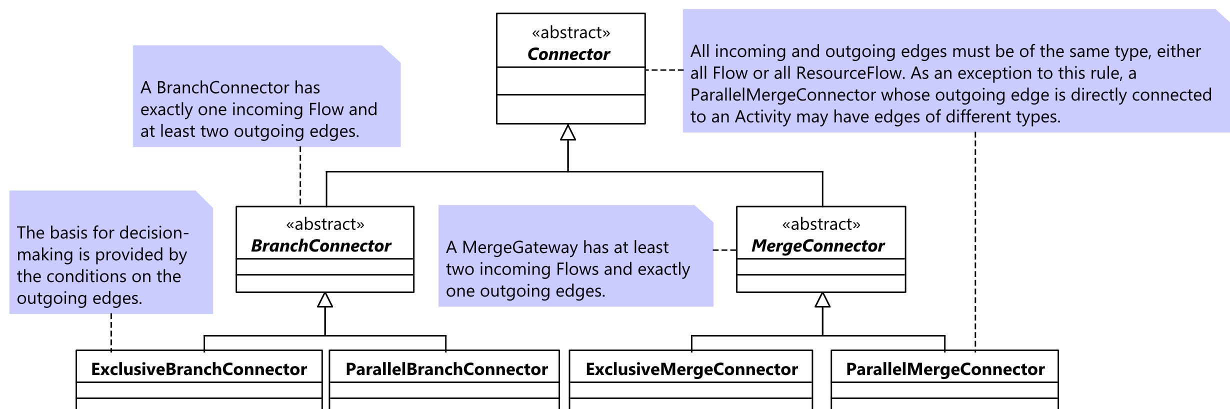


Abbildung 8.20.: Diagramm der **`Connector`**-Klassen im Referenzmetamodell.

8.4. *Resources*

Die in *UML* Aktivitätsdiagrammen, *BPMN* Diagrammen, EPK und Referenzdiagrammen anfallenden Daten werden als Instanzen der *UML* **`Object`**-, *BPMN* **`DataObject`**-, EPK **`InformationObject`**- bzw. **`Resource`**-Klasse repräsentiert und über *UML* **`ObjectFlow`**-, *BPMN* **`SequenceFlow`**-, EPK **`ControlFlow`**- bzw. **`ResourceFlow`**-Klasseninstanzen im Ausführungsfluss transportiert. Instanzen der *UML* **`Object`**-, *BPMN* **`DataObject`**-, EPK **`InformationObject`**- und **`Resource`**-Instanzen erben das Attribut zur Speicherung eines Bezeichners von ihrer jeweiligen **`Elemente`**-Oberklasse. Während dieser Bezeichner bei *UML* **`Object`**- und **`Resource`**-Instanzen für die Speicherung des Instanznamens gedacht ist und über ein ***type***-Attribute zusätzlich der Objekttyp eines Datensatzes gespeichert wird, wird der Bezeichner bei *BPMN* **`DataObject`**- und

EPK ***InformationObject***-Instanzen sowohl zur Speicherung des Instanznamen wie auch für die Speicherung des Typbezeichners genutzt (siehe Lösung der Problemstellung 2. in Unterkapitel 7.3). Im Bezeichner von EPK ***InformationObject***-Instanzen wird zusätzlich noch der Zustand des Datensatzes gespeichert. *UML* ***Object***-, *BPMN* ***DataObject***- und ***Resource***-Instanzen bieten hierfür separate ***state***-Attribute (siehe Lösung der 2. Problemstellung in Unterkapitel 7.3).

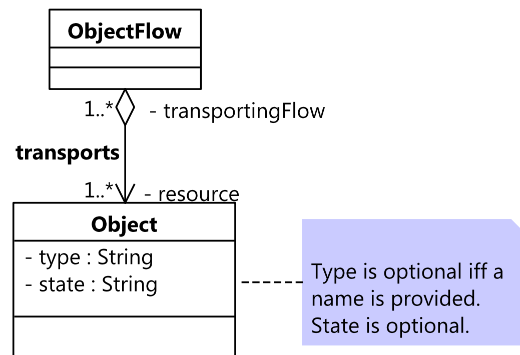


Abbildung 8.21.: Diagramm der *UML* ***Object***-Klasse mit Attributen und Assoziationen.

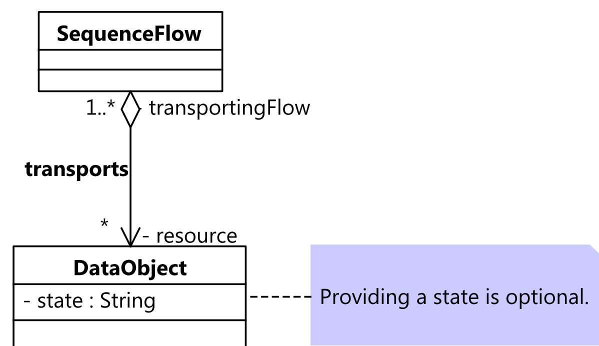


Abbildung 8.22.: Diagramm der *BPMN* ***DataObject***-Klasse mit Attributen und Assoziationen.

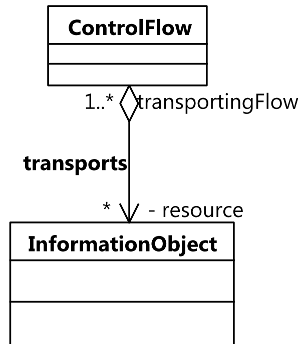


Abbildung 8.23.: Diagramm der EPK **`InformationObject`**-Klasse mit Assoziationen.

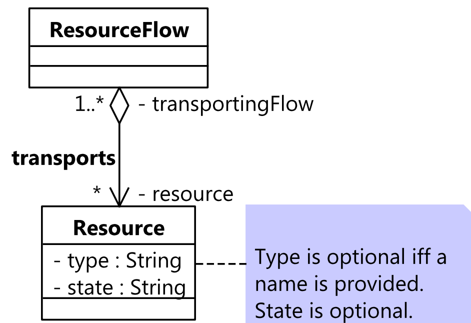


Abbildung 8.24.: Diagramm der **`Resource`**-Klasse mit Attributen und Assoziationen im Referenzmetamodell.

8.5. *Participant Partitions* und Verschachtelungshierarchie

UML-, *BPMN*-, EPK-Modelle und Referenzmodelle bestehen aus einem oder mehreren *UML* Aktivitätsdiagrammen, *BPMN* Diagrammen, EPK bzw. Referenzdiagrammen (siehe Abb. 8.25, 8.26, 8.27 und 8.28).

UML Aktivitätsdiagramme, *BPMN* Diagramme, EPK und Referenzdiagramme besitzen mindestens einen Akteur (***Participant***), welcher die Ausführung der Tätigkeiten im Geschäftsprozess übernimmt. Die Verantwortungsbereiche dieser ***Participants*** werden als Instanzen der *UML* **`ActivityPartition`**-Klasse, der *BPMN* **`Swimlane`**-Unterklassen *BPMN* **`Pool`** und *BPMN* **`Lane`**, der EPK **`OrganizationalUnit`**- bzw. **`ParticipantPartition`**-Klasse dargestellt. *UML* Aktivitätsdiagramme, EPK und Referenzdiagramme besitzen, in Anlehnung an *BPMN* Diagramme, welche immer genau eine *BPMN* **`Pool`**-Instanz haben, immer genau eine oberste (*top-level*) Instanz der *UML* **`ActivityPartition`**-, EPK **`OrganizationalUnit`**- bzw. **`ParticipantPartition`**-Klasse (siehe Lösungen der 1. und 3. Problemstellung in Unterkapitel 7.4). Die Aufteilung des

Verantwortungsbereiches auf weitere Akteure wird in *UML* Aktivitätsdiagrammen, EPK und Referenzdiagrammen als Verschachtelung zusätzlicher Instanzen der *UML* ***ActivityPartition***-, EPK ***OrganizationalUnit*** bzw. ***ParticipantPartition***-Klasse dargestellt. In *BPMN* Diagrammen wird die Aufteilung des Verantwortungsbereiches auf weitere Akteure anhand von *BPMN* ***Lane***-Klasseinstanzen realisiert (siehe Lösung der 2. Problemstellung in Unterkapitel 7.4).

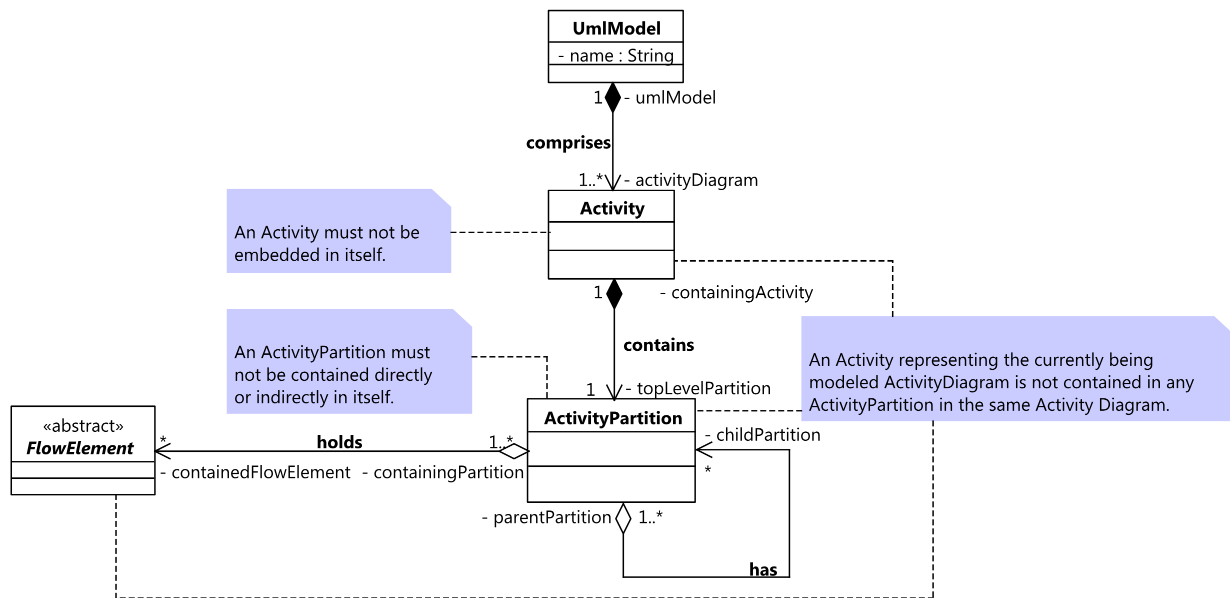


Abbildung 8.25.: Diagramm der Verschachtelungshierarchie von *UML* Aktivitätsdiagrammen.

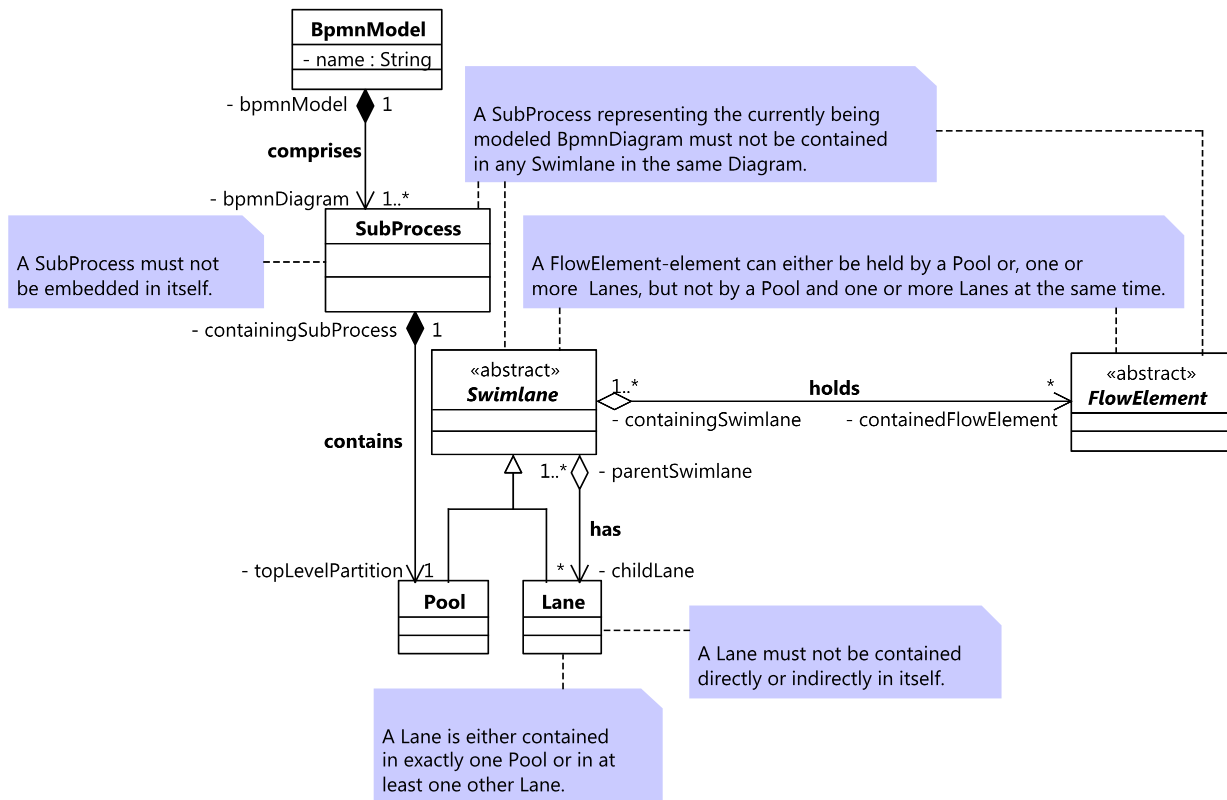


Abbildung 8.26.: Diagramm der Verschachtelungshierarchie von *BPMN* Diagrammen.

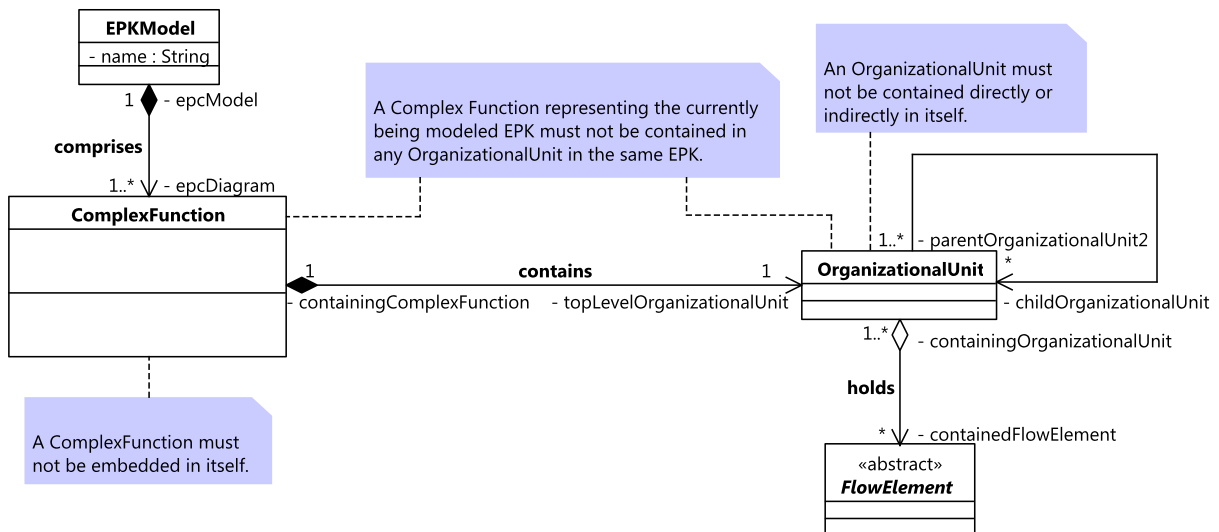


Abbildung 8.27.: Diagramm der Verschachtelungshierarchie von EPK.

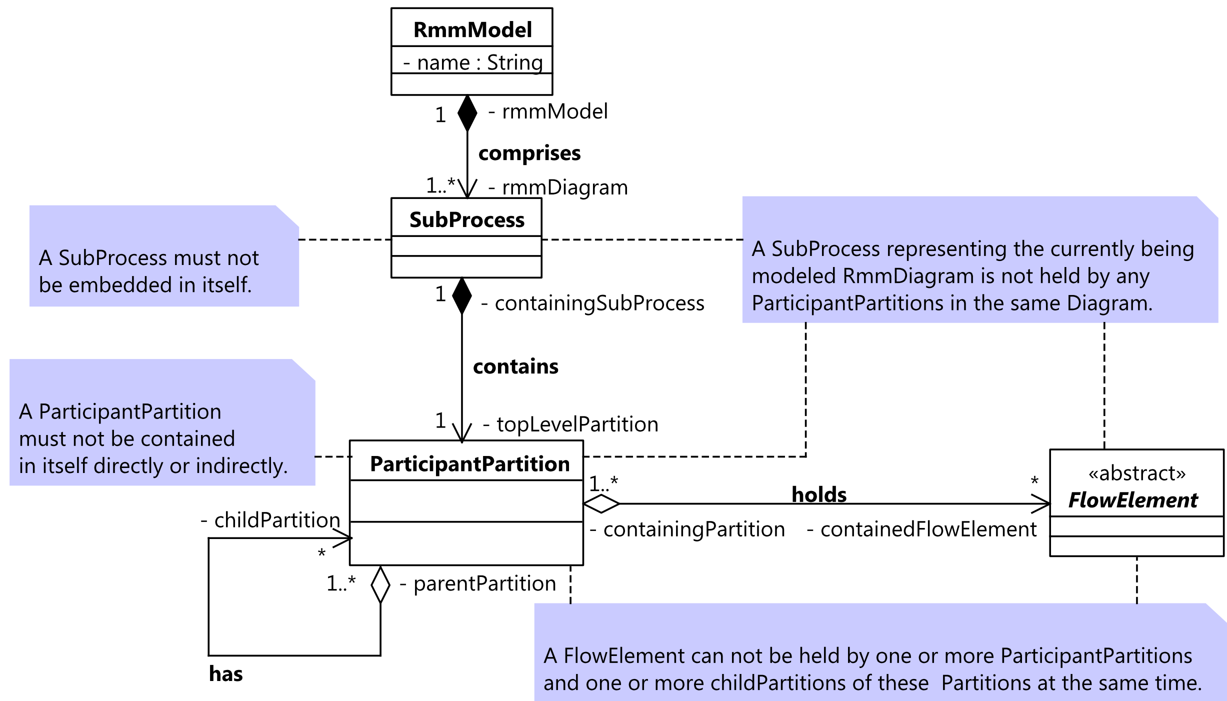


Abbildung 8.28.: Diagramm der Verschachtelungshierarchie in Referenzdiagrammen.

Teil IV.

Exemplarische Realisierung eines Modell-zu-Referenzmodell Konverters für die *BPMN*-Notation

9. Einleitung

Um eine sprach- und werkzeugunabhängige Geschäftsprozessmodellierung zu ermöglichen, wurden zuerst die im *SOAMIG*-Projekt benötigten Sprachkonzepte und -elemente für die Modellierung von Geschäftsprozessen aus den *UML* Aktivitätsdiagrammen, der *BPMN* und der EPK identifiziert und aufeinander abgestimmt behandelt (siehe Teil II und Kapitel 7). Aufbauend auf diesen Konzepten wurden die Sprachmetamodelle und ein Referenzmetamodell entwickelt (siehe Kapitel 8). Auf Basis dieser Metamodelle lassen sich Konvertierer erstellen, welche die Transformation Metamodell-konformer Geschäftsprozessmodelle aus einer Notation in semantisch-äquivalente Modelle der jeweils anderen Sprachen und der Referenz gewährleisten. Anhand dieser Konvertierer gewonnene Referenzmetamodell-konforme Modelle lassen sich schließlich zur Weiterverarbeitung mit *JGraLab* nutzen.

Exemplarisch für die Realisierung von Konvertierungswerkzeugen wird in den folgenden Kapiteln beschrieben, wie Geschäftsprozessmodelle in *BPMN*-Notation in Referenzmetamodell-konforme Modelle überführt werden. Im ersten Schritt wird beschrieben wie Geschäftsprozessmodelle, welche anhand eines beliebigen Modellierungswerkzeugs unter ausschließlicher Nutzung der in dieser Arbeit beschriebenen Sprachkonzepte und Notationselemente, in ein *BPMN*-metamodell-konformes Modell überführt werden (siehe Kapitel 10). Um eine Konvertierung von *BPMN*-Metamodell-konformen Geschäftsprozessmodellen in Referenzmetamodell-konforme Modelle zu ermöglichen, werden anschließend geeignete Transformationsregeln festgelegt (siehe Unterkapitel 11.1) und die Anwendung dieser Regeln anhand eines Konverters beschrieben (siehe Unterkapitel 11.2). Schließlich wird der gesamte Transformationsprozess, vom Ausgangsmodell über das *BPMN*-Metamodell-konforme Modell hin zum Referenzmetamodell-konformen Modell, anhand von Beispielen validiert (siehe Unterkapitel 12).

10. Konvertierung von graphischen *BizAgi*-*BPMN*-Modellen in *BPMN*-Metamodell-konforme Modelle

Vor der Konvertierung eines Geschäftsprozesses aus einer Sprache in eine andere, steht die Erhebung dieser Geschäftsprozesse mit einem Modellierungswerkzeug. Obwohl die Wahl des Modellierungswerkzeugs beliebig ist, ist die Festlegung auf ein bestimmtes Werkzeug, hier zu Demonstrationstzwecken, unumgänglich.

Die Kriterien für die Wahl eines passenden Werkzeugs waren eine möglichst große Abdeckung der in dieser Arbeit beschriebenen Sprachkonzepte und Notationselemente, eine möglichst getreue Einhaltung der *BPMN*-Spezifikation, eine Möglichkeit Modelle in ein semantik-erhaltendes und leicht verarbeitbares Austauschformat zu exportieren sowie die kostenlose Nutzung der Software.

Aus der Liste potentieller Kandidaten wurde das übersichtliche und einfach zu bedienende Modellierungswerkzeug *BizAgi Process Modeler* der Firma *BizAgi Ltd*[1] den gesetzten Kriterien am besten gerecht. Das Programm stellt, nach Umstellung auf einen erweiterten Modus, alle in dieser Arbeit beschrieben Sprachkonzepte und Notationselemente zur Verfügung. Bis auf kleinere Abweichungen, auf die im folgenden Unterkapitel eingegangen wird, setzt das Programm die *BPMN*-Notation spezifikationskonform um. Mit *BizAgi* erstellte *BPMN*-Modelle lassen sich unter Einhaltung der semantischen Äquivalenz in ein *XML*-basiertes Dateiformat – das *XPDL*-Format[2] – exportieren, das in der Industrie weite Verbreitung findet und eine leichte Weiterverarbeitung ermöglicht. Außerdem ist die Nutzung der Software kostenlos.

Um dem Modellierer die Auswahl, der zu den Sprachelementen aus dieser Arbeit passenden, *BizAgi*-Notationselementen zu ermöglichen, wird die Zuordnung der von *BizAgi* angebotenen Notationselemente zu den jeweiligen Sprachelementen aus dieser Arbeit beschrieben (siehe Unterkapitel 10.1). Dann wird die Funktionsweise des Konverters für die Überführung von *BPMN*-Geschäftsprozessmodellen, die im *BizAgi*-Austauschformat vorliegen, in *BPMN*-metamodellkon–forme Modelle beschrieben (siehe Unterkapitel 10.2).

10.1. *Vergleich der Notationselemente aus BizAgi* und dem *BPMN*-Metamodell

Erst mit der Einstellung des erweiterten Modus steht der volle Umfang der *BPMN*-Notationselem–ente in *BizAgi* bereit. Die Zuordnung der *BizAgi* Sprachelemente auf Notationselemente der *BPMN*,

[1] *http://www.bizagi.com/*

[2] http://www.wfmc.org/xpdl.html

welche sich im für diese Arbeit erstellten *BPMN*-Metamodell wiederfinden, sieht wie folgt aus:

- *BizAgi* ***Tasks*** entsprechen *BPMN* ***Tasks***. Für die Spezialisierungen *BizAgi User Task*, *BizAgi Service Task*, *BizAgi Receive Task*, *BizAgi Send Task*, *BizAgi Script Task*, *BizAgi Manual Task* und *BizAgi Reference Task* sind keine Gegenstücke im Metamodell vorgesehen.
- *BizAgi* ***Embedded SubProcesses*** im ausgeklappten (*expanded*) Zustand entsprechen *BPMN* ***Sub-Processes***. Für die Spezialisierung *BizAgi Reusable SubProcess* ist keine Gegenstück im Metamodell vorgesehen.
- *BizAgi* ***Reusable SubProcesses*** entsprechen *BPMN* ***Call Tasks***.
- *BizAgi* ***Start***-Elemente entsprechen *BPMN* ***Start Events***. Für die Spezialisierungen *BizAgi Message Start*, *BizAgi Timer Start*, *BizAgi Conditional Start* und *BizAgi Multiple Start* sind keine Gegenstücke im Metamodell vorgesehen.
- *BizAgi* ***Signal Start***-Elemente und *BizAgi* ***Signal*** **Events** entsprechen *BPMN* ***Signal Catching Events***.
- *BizAgi* ***Signal Events*** bei denen die Option *Is Thrown* augewählt ist und *BizAgi* ***Signal End***-Elemente entsprechen *BPMN* ***Signal Throwing Events***. Für die Spezialisierungen *BizAgi Message Event*, *BizAgi Timer Event*, *BizAgi Compensate Event*, *BizAgi Conditional Event*, *BizAgi Link Event* und *BizAgi Multiple Event* sind keine Gegenstücke im Metamodell vorgesehen.
- *BizAgi* ***End***-Elemente entsprechen *BPMN* ***End Events***.
- *BizAgi* ***Terminate End***-Elemente entsprechen *BPMN* ***Terminate End Events***. Für die Spezialisierungen *BizAgi Message End*, *BizAgi Error End*, *BizAgi Cancel End*, *BizAgi Compensate End* und *BizAgi Multiple End* sind keine Gegenstücke im Metamodell vorgesehen.
- *BizAgi* ***Sequence Flows*** entsprechen *BPMN* ***Sequence Flows***. Für *BizAgi Message Flows* ist kein Gegenstück im Metamodell vorgesehen.

  In BizAgi werden Ausführungsbedingungen (*BPMN* ***Conditions***) für *BizAgi* ***Sequence Flows*** als nicht-graphisches Attribut gespeichert. Um das Ablesen rein graphischer Abbildungen von *BizAgi* Diagrammen vollständig zu ermöglichen, muss der Modellierer daher die Ausführungbedingungen in den Bezeichnern der *BizAgi* ***Sequence Flows*** statt im nicht-graph–ischen Attribut notieren. Da *BPMN* ***Sequence Flow*****s** nicht zwingend einen Namen besitzen müssen ergeben sich folgende Schreibweisen für den Bezeichner von *BizAgi* ***Sequence Flows***:

  - der Bezeichner kann leer sein.
  - der Bezeichner beinhaltet nur den Namen des Ausführungsflusses. Beispiel: *foo.*
  - der Bezeichner beinhaltet nur eine Ausführungsbedingung. Diese wird in eckigen Klammern notiert. Beispiel: *[bar]*.
  - oder der Bezeichner beinhaltet sowohl einen Namen und eine Ausführungsbedingung. Die Ausführungsbedingung wird dann in eckigen Klammern hinter dem Namen notiert. Beispiel: *foo[bar]*.

  Anmerkung: der Bezeichner kann zusätzlich Informationen zu Datensätzen aufweisen (siehe Punkt *BizAgi* ***Data Objects***).

- *BizAgi* ***Data-based Exclusive Gateways*** entsprechen, je nach Anzahl der ein- und ausgehenden Ausführungsflüsse, entweder *BPMN* ***Data-based Exclusive Branch Gateways*** oder *BPMN* ***Data-based Exclusive Merge Gateways***. Für das Notationselement *BizAgi Event-based Exclusive Gateway* ist kein Gegenstück im Metamodell vorgesehen.
- *BizAgi* ***Parallel Gateways*** ensprechen, je nach Anzahl der ein- und ausgehenden Ausführungsflüsse, entweder *BPMN* ***Parallel Branch Gateways*** oder *BPMN* ***Parallel Merge Gateways***. Für die Notationselemente *BizAgi Inclusive Gateway* und *BizAgi Complex Gateway* sind keine Gegenstücke im Metamodell vorgesehen.
- *BizAgi* ***Data Objects*** entsprechen *BPMN* ***Data Objects***. Entgegen der *BPMN*-Spezifikation verbietet *BizAgi* aber die Verbindung von *BizAgi* ***Data Objects*** über *Association*-Kanten mit *BizAgi* ***Sequence Flows***. Der Modellierer muss daher auf die Verwendung von *BizAgi* ***Data Objects*** verzichten und die Informationen zu den Daten in die Bezeichner der Ausführungsflüsse schreiben. Es ergeben sich folgende Schreibregeln:
  - Von einem Objekttyp können ein oder mehrere optionale Instanzen existieren. Die Instanznamen werden untereinander durch Kommata getrennt und, durch einen Doppelpunkt getrennt, vor den obligatorischen Typbezeichner notiert. Für jede Instanz ist zusätzlich die Angabe eines optionalen Objektzustands möglich. Objektzustände werden untereinander durch Kommata getrennt und in eckigen Klammern hinter den Typbezeichner notiert. Die Anzahl der angegebenen Zustände muss dabei mit der Anzahl der Instanznamen übereinstimmen.

    Beispielsweise ist die Notation für die drei Objektinstanzen *a*, *b* und *c* vom Datentyp *x* und den jeweils zugehörigen Zuständen *f*, *g* und *h* wie folgt: *a,b,c:x[f,g,h]*.
  - Werden mehrere Daten von verschiedenen Typen durch ein *BizAgi* ***Sequnce Flow*** transportiert, werden die Informationen der unterschiedlichen Datentypen durch #-Sybol getrennt.

    Beispielsweise ist die Notation für die Objektinstanzen *a,b* und *c* vom Typ *x* und die Instanzen *d*, *e* und *f* vom Typ *y* wie folgt: *a,b,c:x#d,e,f:y*.
  - Die Datenobjekte werden durch die Zeichenkette ## vom restlichen Inhalt des Bezeichners eines *BizAgi* ***Sequence Flows*** notiert. Die Zeichenkette ## wird auch dann vor die Datenobjekte geschrieben, wenn für den BizAgi Sequence Flow kein Name und keine Ausführungsbedingung angegeben wurden.

    Beispielsweise ist die Notation für einen *BizAgi* ***Sequence Flow*** mit dem Namen *foo* und der Ausführungsbedingung *bar*, welcher die Objektinstanzen *a,b* und *c* vom Typ *x* und die Instanzen *d*, *e* und *f* vom Typ *y* mit den jeweiligen Zuständen *g*, *h* und *i* transportiert, wie folgt: *foo[bar]##a,b,c:x#d,e,f:y[g,h,i]*.
- Das *BizAgi* ***Pool***-Element entspricht dem *BPMN* ***Pool***-Element.
- *BizAgi* ***Lanes*** entsprechen *BPMN* ***Lanes***. Im Vergleich zur *BPMN*-Spezifikation ist in *BizAgi* keine Möglichkeit vorgesehen *BizAgi* ***Lanes*** zu verschachteln.
- *BizAgi* ***Annotations*** entsprechen *BPMN* ***Annotations***.
- Für *BizAgi Groups* und *BizAgi Milestones* sind keine Gegenstücke im Metamodell vorgesehen.

10.2. Funktionsweise des Konverters für die Überführung von *BizAgi*-Modellen in *BPMN*-Metamodell-konforme Modelle

Nach der Erstellung eines *BPMN* Modells mit einem oder mehreren *BPMN* Diagrammen in *BizAgi* werden das Modell und alle Diagramme über die *XPDL*-Export Schaltfläche in das *XPDL*-Austauschformat exportiert. Für jedes Modell wird ein Ordner angelegt in den die *XPDL*-Abbilder der Diagramme geschrieben werden. Anhand eines selbst entwickelten *Java*-Programms (siehe Appendix A) wird die Überführung von *BizAgi*-*XPDL*-Modellen in *BPMN*-Metamodell-konforme Modelle durchgeführt.

Programmeingabe: Ein graphisches *BPMN*-Modell das mit dem Modellierungswerkzeug *BizAgi Process Modeler* entwickelt wurde und im *XPDL*-Format vorliegt.

Programmaufruf: *java bpm.jar -i bpmn -t bizagi -o bpmn -p <Ordnerpfad>*

Die Programmoption *-i* (*input*) steht für die Sprache in der das Geschäftsprozessmodell vorliegt. Die Option *-t* steht für das Modellierungswerkzeug (*tool*) mit dem das Geschäftsprozessmodell erstellt wurde. Die Option *-o* (*output*) für das Format in dem die Ausgabe des Modells erfolgen soll und die Option *-p* (*path*) für einen beliebigen Pfad in dem das Modell gespeichert wurde. Eine vollständige Parameterliste findet sich bei Eingabe von: *java bpm.jar -h*.

Das Eingangsmodell liegt im oben genannten Aufruf in *BPMN*-Notation vor, wurde mit dem Modellierungswerkzeug BizAgi erstellt und befindet sich in einem beliebigen Pfad. Es soll in ein *BPMN*-Metamodell-konformes Ausgangsmodell transformiert werden.

Programmausgabe: Sofern keine Fehler auftreten, werden eine *BPMN* ***BpmnModel***-Instanz, welche sich den Namen mit dem Ordner teilt in dem die *XPDL*-Dateien enthalten sind, sowie *BPMN* ***SubProcess***-Instanzen als Abbilder aller sich in diesem Ordner befindenden *XPDL*-Dateien erstellt. Die *BPMN* ***SubProcess***-Instanzen enthalten alle weiteren modellierten Notationselemente.

Die Ausgabe des Programms wird nicht als Datei auf die Festplatte geschrieben, sondern im Speicher als *TGraph* gehalten. Die Ausgabe kann als Eingabe für den modell-zu-referenzmodell Konverter dienen (siehe Unterkapitel 11.2).

11. Transformation *BPMN*-Metamodell-konformer Modelle in Referenzmetamodell-konforme Modelle

Die Überführung *BPMN*-Metamodell-konformer Modelle in Referenzmetamodell-konforme Modelle lässt sich anhand von Transformationsregeln beschreiben (siehe Unterkapitel 11.1), welche durch einen passenden Konverter auf *BPMN*-Metamodell-konforme Modelle angewendet werden können (siehe Unterkapitel 11.2).

11.1. Transformationsregeln

Die folgenden Regeln beschreiben die Überführung von *BPMN* Klasseninstanzen, *BPMN* Attributen und *BPMN* Assoziationen aus *BPMN* Modellen in entsprechende Abbilder in Referenzmodellen. Die Struktur der folgenden Unterkapitel orientiert sich größtenteils an der Struktur aus den Sprachbeschreibungen und den Metamodellbeschreibungen:

- Oberklassen und ***Annotations*** (siehe Unterkapitel 11.1.1),
- ***Activities*** (siehe Unterkapitel 11.1.2),
- ***Flow Controls*** (siehe Unterkapitel 11.1.3),
- ***Resources*** (siehe Unterkapitel 11.1.4) und
- ***Participant Partitions*** (siehe Unterkapitel 11.1.5).

Zur besseren Unterscheidung zwischen Klassen, Attributen und Assoziationen des BPMN-Meta-modells und des Referenzmetamodells, wird dem Namen von *BPMN* Klassen, *BPMN* Attributen und *BPMN* Assoziationen die Zeichenkette *BPMN.* und dem Namen von Klassen, Attributen und Assoziationen aus dem Referenzmetamodell (kurz RMM) die Zeichenkette RMM. vorgestellt.

11.1.1. Oberklassen und *Annotation*

- Die *BPMN.****BpmnModel***-Klasse entspricht der RMM.***RmmModel***-Klasse.

  Der Wert des BPMN.***name***-Attributs wird bei der Überführung übernommen.

  Für jede, über eine *BPMN.****comprises***-Instanz in einer *BPMN.****BpmnModel***-Instanz enthaltene *BPMN.****SubProcess***-Instanz, gibt es eine entsprechende RMM.***comprises***-Instanz und RMM.***SubProcess***-Instanz.

- Die *BPMN*.**`Element`**-Klasse entspricht der RMM.**`Element`**-Klasse.

  Die Werte der Attribute *BPMN*.**`id`** und *BPMN*.**`name`** werden bei der Überführung übernommen.

  Zu jeder *BPMN*.**`Textannotation`**-Instanz, die einer *BPMN*.**`Element`**-Instanz über eine *BPMN*.**`hasAnnotation`**-Instanz zugeteilt ist, gibt es eine entsprechende RMM.**`Annotation`**-Instanz mit gleicher ***id*** und gleichem textuellen Inhalt, welche über eine entsprechende RMM.**`hasAnnoation`**-Instanz mit einer entsprechenden RMM.**`Element`**-Instanz verbunden ist.

- Die *BPMN*.**`TextAnnotation`**-Klasse entspricht der RMM.**`Annotation`**-Klasse.

  Die ***id***, der textuelle Inhalt und die Beziehungen zu annotierten Elementen werden bei der Überführung übernommen.

- Die *BPMN*.**`FlowElement`**-Klasse entspricht der RMM.**`FlowElement`**-Klasse.

  Zu jeder *BPMN*.**`Swimlane`**-Instanz, zu der eine *BPMN*.**`FlowElement`**-Instanz über eine *BPMN*.**`holds`**-Instanz eine Beziehung hat, gibt es jeweilige RMM.**`ParticipantPartition`**-Instanzen, welche über entsprechende RMM.**`holds`**-Instanzen mit entsprechenden RMM.**`FlowElement`**-Abbildern der *BPMN*.**`FlowElement`**-Instanzen verbunden sind.

  Zu jeder *BPMN*.**`SequenceFlow`**-Instanz, zu der eine *BPMN*.**`FlowElement`**-Instanz über eine *BPMN*.**`enters`**-Instanz eine Beziehung hat, gibt es eine entsprechende RMM.**`Flow`**-Instanz, welche über eine entsprechende RMM.**`enters`**-Instanz mit einer entsprechenden RMM.**`FlowElement`**-Instanz verbunden ist.

  Zu jeder *BPMN*.**`SequenceFlow`**-Instanz, zu der eine *BPMN*.**`FlowElement`**-Instanz über eine *BPMN*.**`exits`**-Instanz eine Beziehung hat, gibt es eine entsprechende RMM.**`Flow`**-Instanz, welche über eine entsprechende RMM.**`exits`**-Instanz mit einer entsprechenden RMM.**`FlowElement`**-Instanz verbunden ist.

11.1.2. *Activities*

- Die abstrakte *BPMN*.**`Activity`**-Klasse entspricht der abstrakten RMM.**`Activity`**-Klasse.
  - Die *BPMN*.**`Task`**-Klasse entspricht der RMM.**`AtomicProcess`**-Klasse.
    - Die *BPMN*.**`CallTask`**-Klasse wird auf die RMM.**`CallProcess`**-Klasse abgebildet.

      Für jede *BPMN*.**`invokesBehaviorOf`**-Instanz existiert eine entsprechende RMM.**`invokesBehaviorOf`**-Instanz. Als Abbild der aufgerufenen *BPMN*.**`SubProcess`**-Instanz gibt es eine entsprechende RMM.**`SubProcess`**-Instanz im Referenzmodell.
  - Die *BPMN*.**`SubProcess`**-Klasse entspricht der RMM.**`SubProcess`**-Klasse.

    Zu der über die Instanz der *BPMN*.**`contains`**-Assoziation verbundene *BPMN*.**`Pool`**-Instanz gibt es im Referenzmodell ein entsprechendes Abbild als RMM.**`ParticipantPartition`**-Instanz, welches die oberste Partition im Referenzdiagramm darstellt, d.h.

die RMM.***ParticipantPartition***-Instanz spielt bei keiner Instanz der RMM.***has***-Assoziation, welche diese RMM.***ParticipantPartition***-Instanz mit einer anderen RMM.***ParticipantPartition***-Instanz verbindet, die Rolle der RMM.***subPartition***.

Zu jeder *BPMN*.***invokesBehaviorOf***-Instanz und jeder darüber angebundenen *BPMN*.***CallTask***-Instanz gibt es im Referenzmodell ein Abbild in Form einer entsprechenden RMM.***invokesBehaviorOf***-Instanz sowie einer entsprechenden RMM.***CallProcess***-Instanz.

11.1.3. *Flow Controls*

- Die abstrakte *BPMN*.***FlowControl***-Klasse entspricht der RMM.***FlowControl***-Klasse.

Es wird unterschieden zwischen:

- ***Events*** (siehe Unterkapitel 11.1.3.1),
- ***Flows*** (siehe Unterkapitel 11.1.3.2) und
- ***Connectors*** (siehe Unterkapitel 11.1.3.3).

11.1.3.1. *Events*

- Die abstrakte Klasse *BPMN*.***Event*** wird auf die abstrakte Klasse RMM.***Event*** abgebildet.
  - Die Klasse *BPMN*.***StartEvent*** entspricht der Klasse RMM.***StartEvent***.
  - Zur abstrakten *BPMN*.***SignalEvent***-Klasse gibt es im Referenzmetamodell die abstrakte RMM.***SignalEvent***-Klasse.
    - Die *BPMN*.*SignalThrowingEvent*-Klasse wird auf die RMM.***SignalThrowingEvent***-Klasse abgebildet.

      Für jede *BPMN*.***SignalCatchingEvent***-Instanz, welche über eine *BPMN*.***triggers***-Instanz mit einer *BPMN*.***SignalThrowingEvent***-Instanz verbunden ist, gibt es im Referenzmodell eine entsprechende RMM.***SignalCatchingEvent***-Instanz, welche über eine entsprechende RMM.***triggers***-Instanz mit einer entsprechenden RMM.***SignalThrowingEvent***-Instanz verbunden ist.

      - Die *BPMN*.***ResourceThrowingEvent***-Klasse wird auf die RMM.***ResourceThrowingEvent***-Klasse abgebildet.
    - Die Klasse *BPMN*.***SignalCatchingEvent*** entspricht der RMM.***SignalCatchingEvent***-Klasse.

      Für jede *BPMN*.***SignalThrowingEvent***-Instanz, welche über eine *BPMN*.***triggers***-Instanz mit einer *BPMN*.***SignalCatchingEvent***-Instanz verbunden ist, gibt es im Referenzmodell eine entsprechende RMM.***SignalThrowingEvent***-Instanz, welche über eine entsprechende RMM.***triggers***-Instanz mit einer entsprechenden RMM.***SignalCatchingEvent***-Instanz verbunden ist.

        · Die *BPMN*.***ResourceCatchingEvent***-Klasse wird auf die RMM.***ResourceCatchingEvent***-Klasse abgebildet.

  - Die Klasse *BPMN*.***EndEvent*** entspricht der RMM.***FlowFinalEvent***-Klasse.

    * Die *BPMN*.***TerminateEndEvent***-Klasse wird auf die RMM.***ProcessFinalEvent***-Klasse abgebildet.

11.1.3.2. *Flows*

- Eine *BPMN*.***SequenceFlow***-Instanz entspricht einer RMM.***Flow***-Instanz sofern die *BPMN*.***SequenceFlow***-Instanz nicht über *BPMN*.***transports***-Instanzen mit *BPMN*.***DataObject***-Instanzen verbunden ist, d.h. sofern der Ausführungsfluss keine Daten führt.

  Ist eine *BPMN*.***SequenceFlow***-Instanz hingegen über *BPMN*.***transports***-Instanzen mit *BPMN*.***DataObject***-Instanzen verbunden, wird die *BPMN*.***SequenceFlow***-Instanz im Referenzmodell als RMM.***ResourceFlow***-Instanz abgebildet. Für jede *BPMN*.***DataObject***-Instanz, zu der eine *BPMN*.***SequenceFlow***-Instanz über eine *BPMN*.***transports***-Instanz eine Verbindung hat, existiert im Referenzmodell eine entsprechende RMM.***Resource***-Instanz, welche über eine entsprechende RMM.***transports***-Instanz mit der entsprechenden RMM.***ResourceFlow***-Instanz verbunden ist.

  Wurde dem *BPMN*.***condition***-Attribut einer *BPMN*.***SequenceFlow***-Instanz kein Wert zugeteilt (also *null*), entspricht der Wert des RMM.***guard***-Attributs der entsprechenden RMM.***Flow***-Instanz dem boolschen Wahrheitswert *true*. Ansonsten wird der Wert der Variable übernommen.

11.1.3.3. *Connectors*

- Die abstrakte *BPMN*.***Gateway***-Klasse entspricht der abstrakten RMM.***Connector***-Klasse.

  - Die abstrakte *BPMN*.***BranchGateway***-Klasse wird auf die RMM.***BranchConnector***-Klasse abgebildet.

    * Die *BPMN*.***DataBasedExclusiveBranchGateway***-Klasse entspricht der RMM.***ExclusiveBranchConnector***-Klasse.

    * Die *BPMN*.***ParallelBranchGateway***-Klasse entspricht der RMM.***ParallelBranchConnector***-Klasse.

  - Die abstrakte *BPMN*.***MergeGateway***-Klasse wird auf die RMM.**Merge*****Connector***-Klasse abgebildet.

    * Die *BPMN*.***DataBasedExclusiveMergeGateway***-Klasse entspricht der RMM.***ExclusiveMergeConnector***-Klasse.

    * Die *BPMN*.***ParallelMergeGateway***-Klasse entspricht der RMM.***ParallelMergeConnector***-Klasse.

11.1.4. *Resources*

- Die *BPMN*.***DataObject***-Klasse entspricht der RMM.***Resource***-Klasse.

  Enthält die Zeichenkette des *BPMN*.***name***-Attributs einer *BPMN*.***DataObject***-Instanz einen Doppelpunkt, wird die Zeichenkette hinter dem Doppelpunkt, bei der Überführung ins Referenzmodell, dem RMM.***type***-Attribut einer entsprechenden RMM.***Resource***-Instanz zugewiesen. Die Zeichenkette vor dem Doppelpunkt wird der RMM.***name***-Variable der entsprechenden RMM.***Resource***-Instanz zugewiesen. Der Doppelpunkt wird nicht übernommen.

  Enthält die Zeichenkette des *BPMN*.***name***-Attributs einer *BPMN*.***DataObject***-Instanz hingegen keinen Doppelpunkt, wird dem RMM.***type***-Attribut einer entsprechenden RMM.***Resource***-Instanz kein Wert zugewiesen (also *null*). Die Zeichenkette des *BPMN*.***name***-Attributs wird komplett als Wert des RMM.***name***-Attributs in der entsprechenden RMM.***Resource***-Instanz abgespeichert.

  Der Wert des *BPMN*.***state***-Attributs einer *BPMN*.***DataObject***-Instanz wird als Wert für das RMM.***state***-Attribut einer entsprechenden RMM.***DataObject***-Instanz übernommen.

  Zu jeder *BPMN*.***SequenceFlow***-Instanz, welche über *BPMN*.***transports***-Instanzen mit BPMN.***DataObject***-Instanzen verbunden ist, existieren im Referenzmodell entsprechende RMM.***ResourceFlow***-Instanzen, welche über RMM.***transports***-Instanzen mit entsprechenden RMM.***Resource***-Instanzen verbunden sind.

11.1.5. *Participant Partitions*

- Ein direktes Abbild der *BPMN*.***Swimlane***-Klasse im Referenzmetamodell ist nicht vorgesehen, stattdessen ist die Abbildung von der Unterklasse abhängig.

  – Die *BPMN*.***Pool***-Klasse entspricht der Klasse RMM.***ParticipantPartition***, mit der Einschränkung, dass das RMM.***ParticipantPartition***-Abbild einer *BPMN*.***Pool***-Instanz im Referenmodell keine Instanz der RMM.***has***-Assoziation besitzt in der diese RMM.***ParticipantPartition***-Instanz die Rolle als RMM.***childPartition*** einnimmt, d.h. die RMM.**PareticipantPartition**-Instanz ist die oberste (*top-level*) Partition im Referenzdiagramm.

    Besitzt eine *BPMN*.***Pool***-Instanz eine Verbindung zu *BPMN*.***Lane***-Instanzen über *BPMN*.***has***-Assoziationen, so ist das entsprechende RMM.***ParticipantPartition***-Abbild der *BPMN*.***Pool***-Instanz im Referenzmodell über entsprechende Instanzen der RMM.***has***-Assoziation mit entsprechenden RMM.***ParticipantPartition***-Abbildern der *BPMN*.***Lane***-Instanzen verbunden.

    Zu der *BPMN*.***SubProcess***-Instanz, zu der eine *BPMN*.***Pool***-Instanz eine *BPMN*.***contains***-Instanz besitzt, existiert im Referenzmodell eine entsprechende RMM.***SubProcess***-Instanz, welche über eine RMM.***contains***-Instanz mit einer entsprechenden RMM.***ParticipantPartition***-Instanz verbunden ist.

– Die *BPMN*.***Lane***-Klasse entspricht der RMM.***ParticipantPartition***-Klasse, mit der Einschränkung, dass das RMM.***ParticipantPartition***-Abbild dieser *BPMN*.***Lane***-Instanz im Referenzmodell mindestens eine Instanz der RMM.***has***-Assoziation besitzt, in der die entsprechende RMM.***ParticipantPartition***-Instanz die Rolle der RMM.***childParition*** einnimmt, d.h. die Partition ist nicht die oberste (*top-level*) im Diagramm.

11.2. Anwendung der Transformationsregeln anhand eines modell-zu-referenzmodell Konverters

Anhand der beschriebenen Regeln lässt sich ein entsprechender Konvertierer schreiben, welcher diese Regeln auf ein *BPMN*-Metamodell-konformes Modell anwendet und ein Referenzmetamodell-konformes Modell liefert. Als exemplarische Realisierung eines solchen Konverters wurde ein *Java*-Programm entwickelt (siehe Appendix A).

Programmeingabe: Ein graphisches *BPMN*-Modell das mit einem beliebigen Modellierungswerkzeug erstellt wurde.

Programmaufruf: *java bpm.jar -i bpmn -t <Modellierungswerkzeug> -o rmm <Ordnerpfad>*

Die Programmoption *-i* (*input*) steht für die Sprache in der das Geschäftsprozessmodell vorliegt. Die Option *-t* steht für das Modellierungswerkzeug (*tool*) mit dem das Geschäftsprozessmodell erstellt wurde. Die Option *-o* (*output*) für das Format in dem die Ausgabe des Modells erfolgen soll und die Option *-p* (*path*) für einen beliebigen Pfad in dem das Modell gespeichert wurde. Eine vollständige Parameterliste findet sich bei Eingabe von: *java bpm.jar -h*.

Das Eingangsmodell liegt im oben genannten Aufruf in *BPMN*-Notation vor, wurde mit einem beliebigen Modellierungswerkzeug entwickelt und liegt in einem beliebigen Ordner. Es soll in ein Referenzmetamodell-konformes Ausgangsmodell transformiert werden.

Programmausgabe: Sofern keine Fehler auftreten, werden den Regeln entsprechende Transformationen des *BPMN*-Modells vorgenommen und ein Referenzmetamodell-konformes Modell zurückgegeben. Dieses Modell stellt die Basis für weiterführende Tätigkeiten im Kontext des *SOAMIG*-Projekts dar.

Die Ausgabe des Programms wird nicht als Datei auf die Festplatte geschrieben, sondern im Speicher als *TGraph* gehalten.

12. Validierung des Ansatzes anhand von Beispielen

In diesem Kapitel werden einige Beispiele zu zentralen Aspekten beschrieben, welche eine große Rolle bei der Abstimmung der Notationssprachen aufeinander, und der Erstellung der Referenz (siehe Unterkapitel 7) spielen. Die Beispieldiagramme wurden in *BPMN*-Notation mit *BizAgi* erstellt und anhand der Konverter in ein *BPMN*-Metamodell-konformes und ein Referenzmetamodell-konformes Modell transformiert. Zur besseren Visualisierung der erzeugten Modelle, werden bei der Transformation mit den Konvertern durch anhängen des Kommandozeilenparameters *-d*, Dateien erzeugt, welche sich mit dem Programm *GraphViz*[1] öffnen lassen und das konvertierte Modell als Graphen dargestellen.

Bei den Beispielen handelt es sich um:

- ein *BPMN* Diagramm mit einer *BPMN* ***Activity***, welche mehrere ein- und ausgehende *BPMN* ***Sequence Flows*** aufweist (siehe Unterkapitel 12.1),
- ein *BPMN* Diagramm mit einem *BPMN* ***Sequence Flow***, welcher mehrere *BPMN* ***Data Objects*** transportiert (siehe Unterkapitel 12.2),
- den Einsatz von *BPMN* ***Signal Throwing Events*** und *BPMN* ***Signal Catching Events*** sowie *BPMN* ***Resource Throwing Events*** und *BPMN* ***Resource Catching Events*** in *BPMN* Diagrammen (siehe Unterkapitel 12.3),
- den Einsatz von *BPMN* ***Pools*** in *BPMN* Diagrammen (siehe Unterkapitel 12.4), und
- die Darstellung des Geschäftsprozesses *sellTrainticket*, welcher als Einführungsbeispiel in den Kapiteln zu den Sprachbeschreibungen diente (siehe Unterkapitel 3.2, 4.2 und 5.2), als *BizAgi BPMN* Diagramm und der anschließenden Konvertierung in *BPMN*-Metamodell-konforme und Referenzmetamodell-konforme Modelle (siehe Unterkapitel 12.5).

12.1. Beispiel: Multiple ein- & ausgehende Ausführungsflüsse

Bei der Überführung eines *BPMN* Diagrammes/ *BPMN*-Metamodell-konformen Modells, in dem mehrere Instanzen der *BPMN.*`SequenceFlow`-Klasse in eine Instanz der *BPMN*.`Activity`-Klasse eingehen (siehe Abb. 12.1), in ein Referenzmetamodell-konformes Modell, ist die Erzeugung einer zusätzlichen RMM.`MergeConnector`-Instanz für die Zusammenführung der Abbilder der Ausführungsflüsse vor der RMM.`Activity`-Instanz nicht nötig, da mehrere in eine RMM-`Activity`-Instanz eingehende Ausführungflüsse, wie auch mehrere in eine *BPMN*-`Activity`-Instanz eingehende Ausführungsflüsse, eine implizite *OR*-Verknüpfung darstellen.

[1] `http://www.graphviz.org/`

Mehrere augehende Ausführungsflüsse bilden dagegen, sowohl in *BPMN* Diagrammen wie in Referenzdiagrammen, eine *AND*-Aufteilung. Auch hier ist die Erzeugung einer zusätzlichen RMM.***BranchConnector***-Instanz bei der Konvertierung überflüssig.

Als Konsequenz sind die *BPMN* ***Sequence Flows*** *a* und *b*, im *BPMN*-Metamodell-konformen Modell (siehe Abb. 12.2), als *BPMN*.***SequenceFlow***-Instanzen mit den *GraphViz*-intern vergebenen Bezeichnern *v10* und *v11*, direkt über je eine Instanz der *BPMN*.***enters***-Assoziation – *e10* bzw. *e12* – mit der *BPMN*.***Task***-Instanz, mit dem Bezeichner *v6*, verbunden. Die *BPMN*.***SequenceFlow***-Instanzen mit Namen *c* und *d* – *v9* bzw. *v12* – sind jeweils über eine Instanz der *BPMN*.***exits***-Assoziation – *e9* bzw. *e15* – direkt mit der *BPMN*.***Task***-Instanz *v6* verbunden.

Im Referenzmetamodell-konformen Modell (siehe Abb. 12.3), welches aus dem *BPMN*-metamodell-konformen Modell erzeugt wurde, sind die RMM.***Flow***-Instanzen mit den Namen *a* und *b* – *v10* bzw. *v11* – direkt über je eine Instanz der RMM.***enters***-Assoziation – *e13* bzw. *e14* – mit der RMM.***AtomicProcess***-Instanz mit dem Bezeichner *v* verbunden. Die RMM.***Flow***-Instanzen mit den Namen *c* und *d* – *v9* bzw. *v12* – sind jeweils über eine Instanz der RMM.***exits***-Assoziation – *e8* bzw. *e11* – mit der RMM.***AtomicProcess***-Instanz *v6* verbunden.

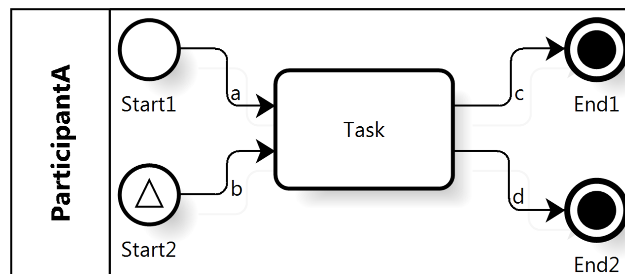


Abbildung 12.1.: *BPMN* Diagramm in *BizAgi*-Notation, welches eine *BPMN* ***Activity*** mit mehreren ein- und ausgehenden *BPMN* ***Sequence Flows*** enthält.

Abbildung 12.2.: Graphische Repräsentation mit *GraphViz* der, durch den Konverter erstellten, Knoten- und Kanteninstanzen im *BPMN*-Metamodell-konformen Modell.

Abbildung 12.3.: Graphische Repräsentation mit *GraphViz* der, durch den Konverter erstellten, Knoten- und Kanteninstanzen im Referenzmetamodell-konformen Modell.

12.2. Beispiel: Transport mehrerer Datensätze

Aufgrund einer Limitierung seitens der Modellierungssoftware *BizAgi*, können *BPMN* ***Data Objects*** nicht, wie in der *BPMN*-Spezifikation beschrieben, direkt mit *BPMN* ***Sequence Flows*** assoziiert werden (siehe Abb. 12.4). Stattdessen muss der Modellierer die von einem BPMN Sequenece Flow zu transportierenden Datensätze im Bezeichner dieses Flusses notieren (siehe Abb. 12.5).

Bei der Abbildung auf ein *BPMN*-Metamodell-konformes Modell werden die Bezeichner von Instanzen der *BPMN*.***SequenceFlow***-Klasse analysiert und ausgemachte Datensätze als Instanzen der *BPMN*.***DataObject***-Klasse im Modell notiert. Da die *BPMN*-Spezifikation keine Unterscheidung zwischen Instanznamen und Objekttyp für *BPMN* ***Data Objects*** vorsieht, werden Instanznamen und Objekttyp gemeinsam im *BPMN*.***name***-Attribut der jeweiligen *BPMN*.***DataObject***-Instanzen gespeichert (siehe Knoteninstanzen *v9*, *v10* und *v11* in Abb. 12.6).

Bei der Abbildung auf ein Referenzmetamodell-konformes Modell werden alle *BPMN*.***DataObject***-Instanzen in entsprechende Instanzen der RMM.***Resource***-Klasse transformiert. Dabei wird der Instanzname aus dem *BPMN*.***name***-Attribut im RMM.***name***-Attribut und der Objekttyp aus dem *BPMN*.***name***-Attribut im RMM.***type***-Attribut gespeichert (siehe Knoteninstanzen *v9*, *v10* und *v11* in Abb. 12.7).

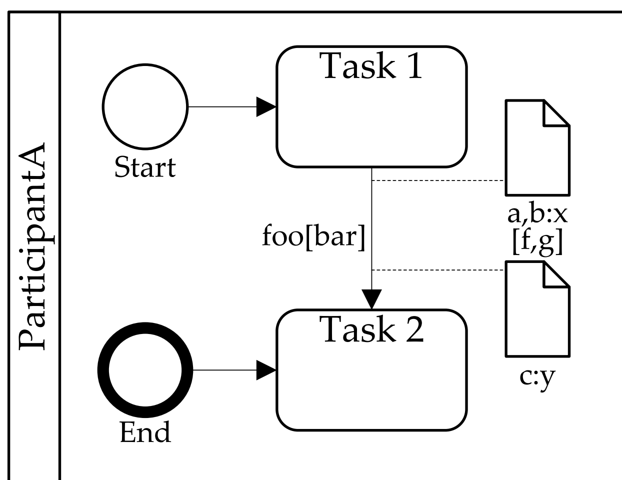


Abbildung 12.4.: *BPMN*-spezifikationskonforme direkte Assoziierung von *BPMN* ***Data Objects*** an einen *BPMN* ***Sequence Flow***.

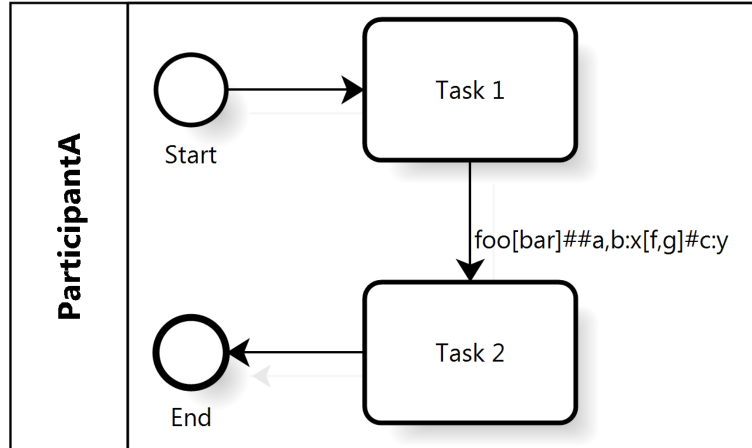


Abbildung 12.5.: Durch eine Limitierung in BizAgi können *BPMN* ***Data Objects*** nicht direkt mit einem *BPMN* ***Sequence Flow*** assoziiert werden. Stattdessen müssen die Informationen zu den Datensätzen im Bezeichner des Ausführungsflusses notiert werden.

Abbildung 12.6.: Darstellung des Sachverhalts als *BPMN*-Metamodell-konformes Modell, nach der Konvertierung aus dem *BizAgi* Modell. Die Datensätze wurden aus dem Bezeichner des *BPMN* ***Sequence Flows*** extrahiert und werden im Modell als Instanzen der *BPMN*.**`DataObject`**-Klasse dargestellt – Knoteninstanzen *v9*, *v10* und *v11*. Die Instanzen sind jeweils über eine Instanz der *BPMN*.**`transports`**-Assoziation – Kanteninstanzen *e9*, *e10* und *e11* – mit der *BPMN*.**`SequenceFlow`**-Instanz *v8* verbunden, welche die Datensätze transportiert.

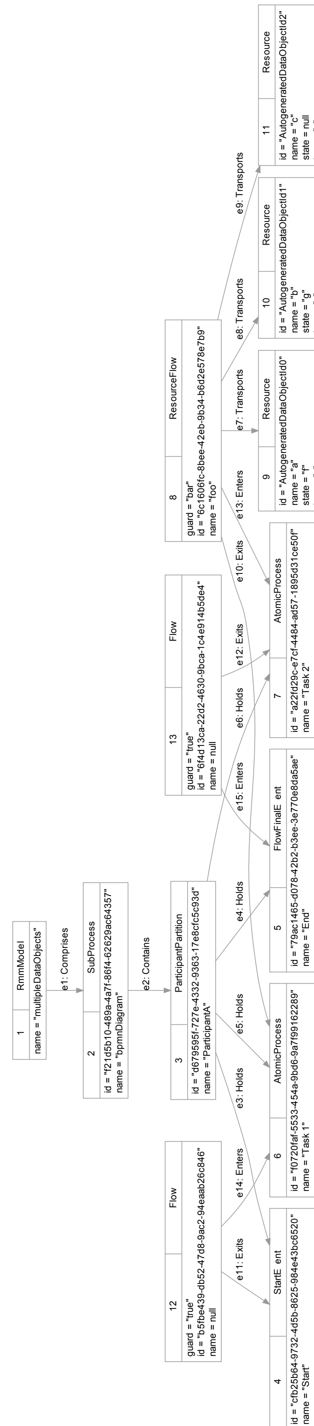


Abbildung 12.7.: Darstellung des Sachverhalts als Referenzmetamodell-konformes Modell, nach der Konvertierung aus dem *BPMN*-Metamodell-konformen Modell. Die Datensätze werden im Modell als Instanzen der RMM.***Resource***-Klasse dargestellt – Knoteninstanzen *v9*, *v10* und *v11*. Die Instanzen sind jeweils über eine Instanz der RMM.***transports***-Assoziation – Kanteninstanzen *e7*, *e8* und *e9* – mit der RMM.***ResourceFlow***-Instanz *v8* verbunden, welche die Datensätze transportiert.

12.3. Beispiel: Einsatz von *BPMN Signal & Resource Events*

Resource Throwing Events und ***Resource Catching Events*** unterscheiden sich von ***Signal Throwing Events*** und ***Signal Catching Events*** nicht nur durch den Namenszusatz <<*Resource Throwing Event*>> bzw. <<*Resource Catching Event*>>, oder alternativ durch ein nachgestelltes #-Symbol im Namen, sondern auch dadurch, dass ***Resource Throwing Events*** welche namentlich zugehörige ***Resource Catching Events*** auslösen in verschieden Diagrammen/ Unterprozessen stehen müssen, während zusammengehörige ***Signal Throwing Events*** und ***Signal Catching Events*** im gleichen Diagramm / Unterprozess stehen dürfen (siehe Abb. 12.8).

Bei der Abbildung in ein *BPMN*-Metamodell-konformes Modell werden namentlich passende Instanzen der *BPMN*.***SignalThrowingEvent***- und *BPMN*.***SignalCatchingEvent***-Klasse – Knoteninstanzen *v17* und *v11* – über Instanzen der *BPMN*.***trigger***-Assoziation – *e16* – miteinander verbunden. Namentlich passende Instanzen der *BPMN*.***ResourceThrowingEvent***- und *BPMN*.***ResourceThrowingEvent***-Klasse – *v18* und *v10* – werden, sofern sie sich in verschiedenen *BPMN* Diagramm/ *BPMN* ***SubProcess***-Instanzen befinden – *v2* enthält *v18* und v6 enthält *v10* – über Instanzen der *BPMN*.***trigger***-Assoziation – *e18* – verbunden, andernfalls bricht der Konverter die Transformation mit einer Fehlermeldung ab.

Sind passende Instanzen der *BPMN*.***ResourceThrowingEvent***- und *BPMN*.***ResourceCatchingEvent***-Klasse miteinander über Instanzen der *BPMN*.***trigger***-Assoziation verbunden, wird für jeden Datensatz, welcher über eine *BPMN*.***SequenceFlow***-Instanz, die in eine *BPMN*.***ResourceThrowingEvent***-Instanz eingeht und über eine oder mehrere *BPMN*.***SequenceFlow***-Instanzen, die aus namentlich zugehörigen *BPMN*.***ResourceCatchingEvent***-Instanzen ausgehen, transportiert, wird genau ein Abbild als *BPMN*.***DataObject***-Instanz im Modell erstellt und über entsprechende Instanzen der *BPMN*.***transports***-Assoziation mit den entsprechenden *BPMN*.***SequenceFlow***-Instanzen verbunden. Angewandt auf ein *BPMN* Diagramm ergibt sich, für das *BPMN* ***Data Object*** mit dem Namen *a:x* und Zustand *[f]*, bei der Überführung in ein *BPMN*-Metamodell-konformes Modell genau eine *BPMN*.***DataObject***-Instanz – *v21*. Diese Instanz wird über Instanzen der *BPMN*.***transports***-Assoziation – *e31* bzw. *e24* – mit den *BPMN*.***SequenceFlow***-Instanzen *v24* und *v20* verknüpft, welche den Datensatz transportieren (siehe Abb. 12.9).

Die Abbildung in ein Referenzmetamodell-konformes Modell erfolgt analog (siehe Abb. 12.10).

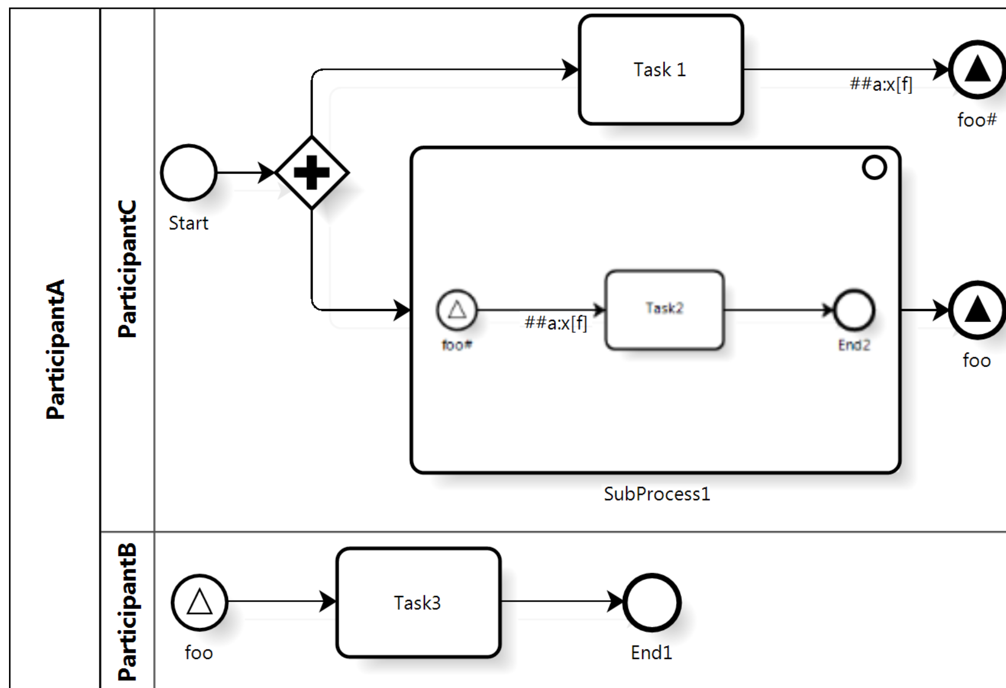


Abbildung 12.8.: Abbildung des Sachverhalts als *BizAgi* Diagramm.

Abbildung 12.9.: Überführung des Sachverhalts aus einem BizAgi Diagramm in ein BPMN-Metamodell-konformes Modell.

Abbildung 12.10.: Darstellung des Sachverhalts als Referenzmetamodell-konformes Modell, nach der Konvertierung aus dem *BPMN*-Metamodell-konformen Modell.

12.4. Beispiel: Einsatz von *BPMN Pools*

Nach der BPMN-Spezifikation muss ein *BPMN* Diagramm/ *BPMN* ***Sub-Process*** immer mindestens einen *BPMN* ***Pool*** enthalten:

- Wird ein *BPMN* ***Pool*** in einem Diagramm explizit modelliert (siehe Abb. 12.11), dann wird als dessen Abbild eine *BPMN*.***Pool***-Instanz im *BPMN*-Metamodell-konformen Modell (siehe Abb. 12.12) bzw. eine entsprechende RMM.***ParticipantPartition***-Instanz im Referenzmetamodell-konformen Modell erstellt (siehe Abb. 12.13).

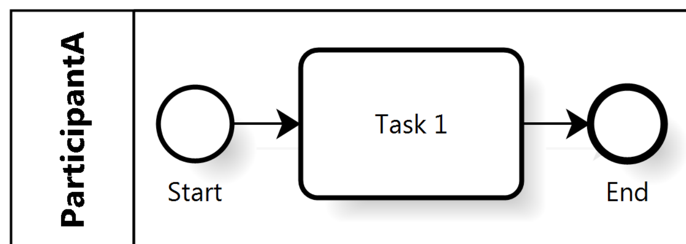


Abbildung 12.11.: Ein *BizAgi* Diagramm mit einem explizit modellierten *BPMN* ***Pool***, namens *ParticipantA*.

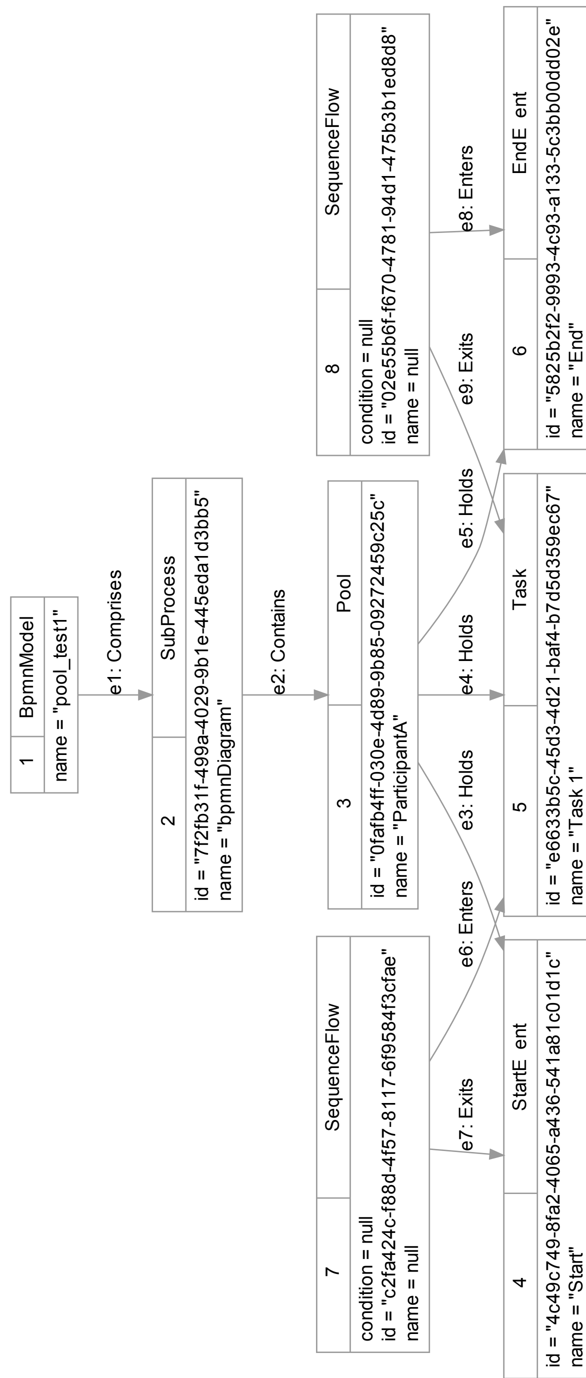


Abbildung 12.12.: Bei der Überführung in ein *BPMN*-Metamodell-konformes Modell wird eine *BPMN*.***Pool***-Instanz *v3*, als Abbild für den explizit modellierten *BPMN* ***Pool*** mit Namen *ParticipantA*, erstellt.

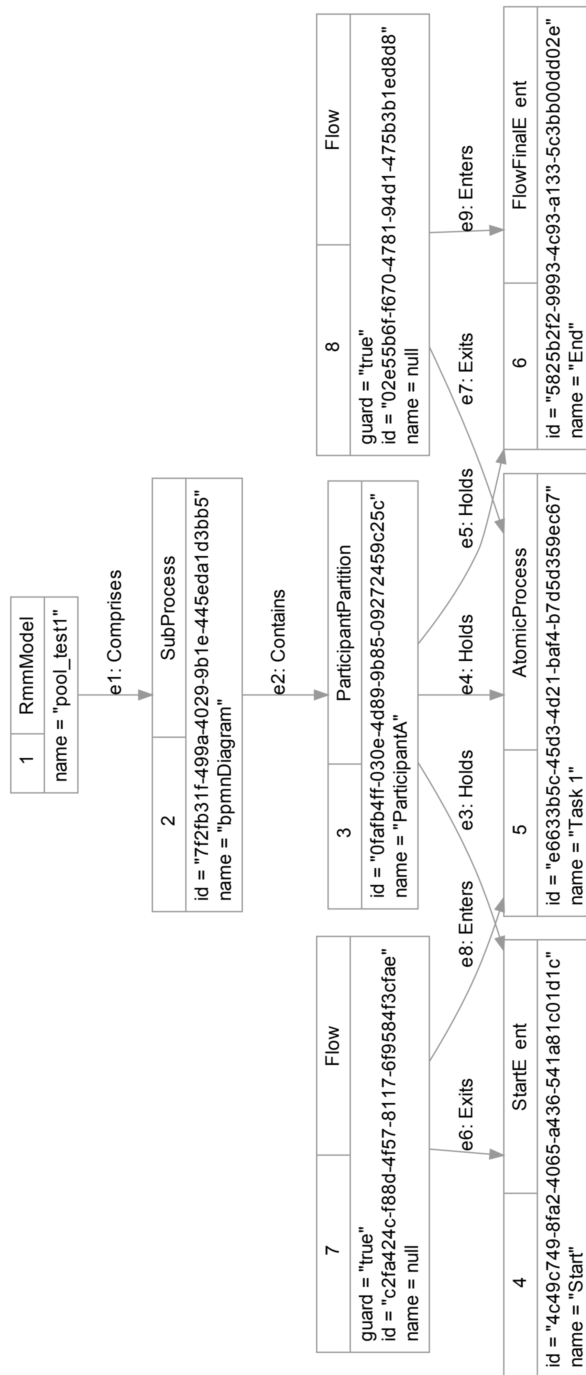


Abbildung 12.13.: Bei der Überführung in ein Referenzmetamodell-konformes Modell wird eine RMM.***`ParticipantPartition`***-Instanz *v3*, als Abbild der *BPMN.**`Pool`***-Instanz mit dem Namen *ParticipantA*, erstellt.

- Wird hingegen explizit kein *BPMN* ***Pool*** modelliert (siehe Abb. 12.14), enthält das *BPMN* Diagramm trotzdem einen *BPMN* ***Pool***, welcher nicht graphisch dargestellt wird. Dieser unsichtbare *BPMN* ***Pool*** wird bei der Überführung in ein *BPMN*-Metamodell-konformes Modell als *BPMN*.***Pool***-Instanz dargestellt, welche sich den Namen mit dem *BPMN* Diagramm, in dem sie sich befindet, teilt und eine, bei der Transformation automatisch generierte, *id* erhält (siehe Abb. 12.15). Bei der Transformation in ein Referenzmetamodell-konformes wird diese *BPMN*.***Pool***-Instanz regulär, wie ein explizit modellierter *BPMN* ***Pool***, in eine RMM.***ParticipantPartition***-Instanz transformiert (siehe Abb. 12.16).

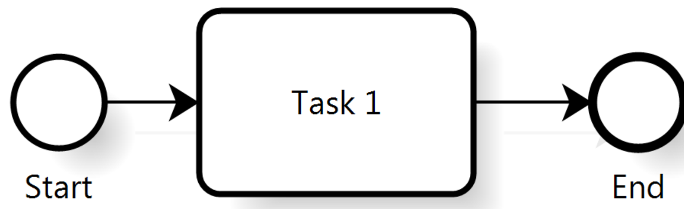


Abbildung 12.14.: Ein *BPMN* Diagramm ohne explizit modellierten *BPMN* ***Pool***.

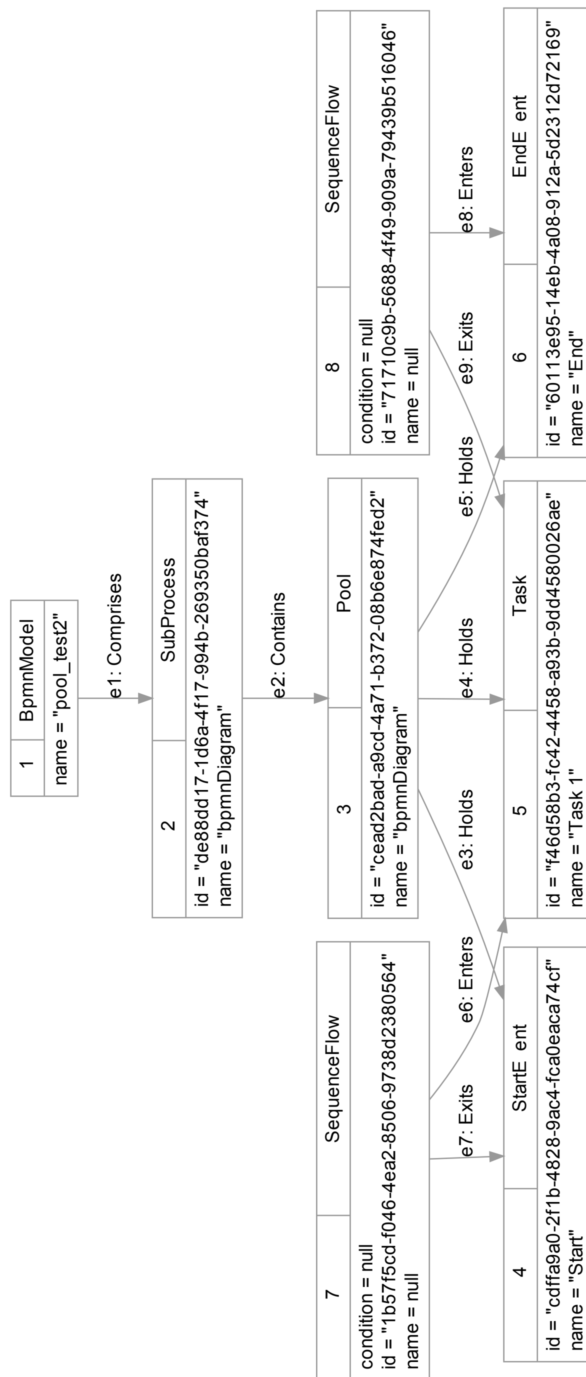


Abbildung 12.15.: Bei der Überführung in ein *BPMN*-Metamodell-konformes Modell tritt der nicht-graphische *BPMN* ***Pool*** als *BPMN*.***`Pool`***-Instanz in Erscheinung – Knoteninstanz *v3*. Der Name wird dabei von der sie enthaltenden *BPMN*.***`SubProcess`***-Instanz übernommen, die *id* wird bei der Konvertierung automatisch generiert.

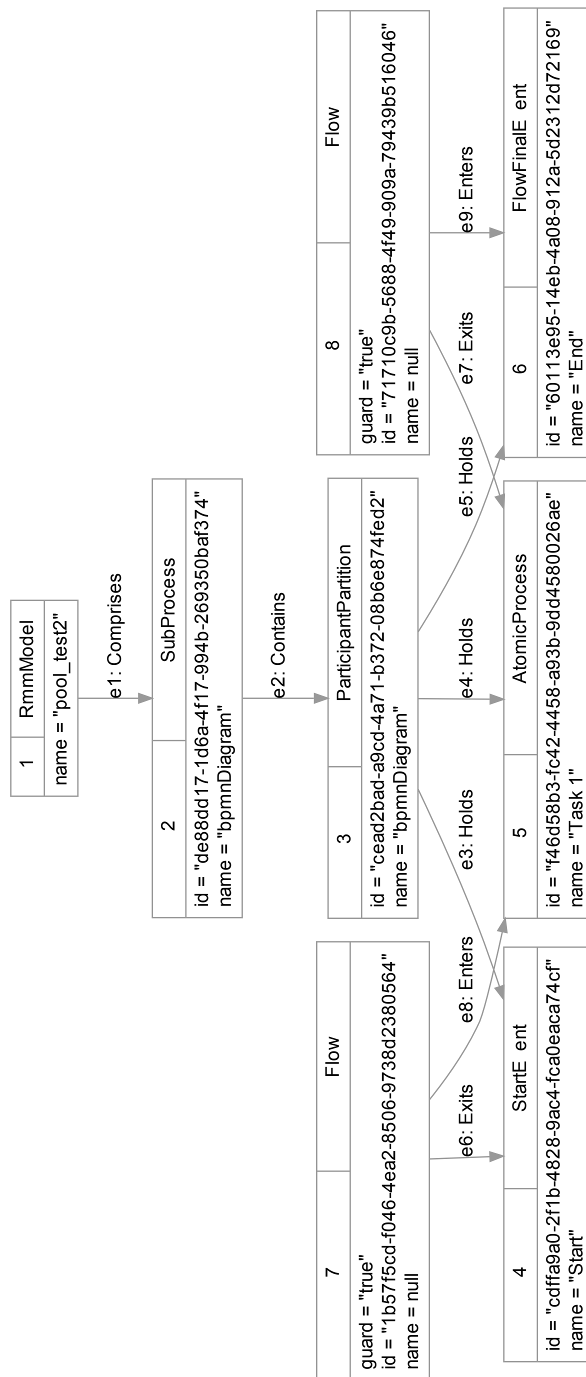


Abbildung 12.16.: Bei der Überführung des *BPMN*-Metamodell-konformen Modells in ein Referenzmetamodell-konformes Modell wird, wie für einen explizit modellierten *BPMN* ***Pool***, eine entsprechende RMM.***ParticipantPartition***-Instanz erstellt – Knoteninstanz *v3*.

- Einem *BPMN* ***Sub-Process***, welcher in *BPMN* Diagramm eingebettet wird, kann explizit kein *BPMN* ***Pool*** zugewiesen werden (siehe Abb. 12.17). Trotzdem enthält jeder *BPMN* ***Sub-Process*** einen nicht-graphischen *BPMN* ***Pool***, welcher bei der Überführung in ein *BPMN*-Metamodell-konformes Modell als *BPMN*.***`Pool`***-Instanz (siehe Abb. 12.18) und in ein Referenzmetamodell-konformes Modell als RMM.***`ParticipantPartition`***-Instanz (siehe Abb. 12.19) dargestellt wird.

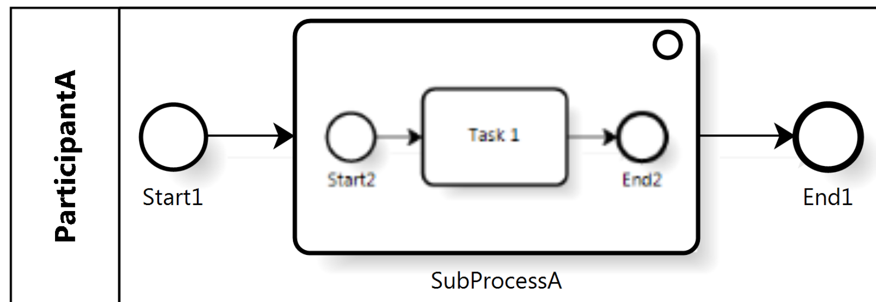


Abbildung 12.17.: Einem *BPMN* ***Sub-Process*** lässt sich explizit kein *BPMN* ***Pool*** zuweisen. Trotzdem enthält jeder *BPMN* ***Sub-Process*** einen nicht-graphischen *BPMN* ***Pool***.

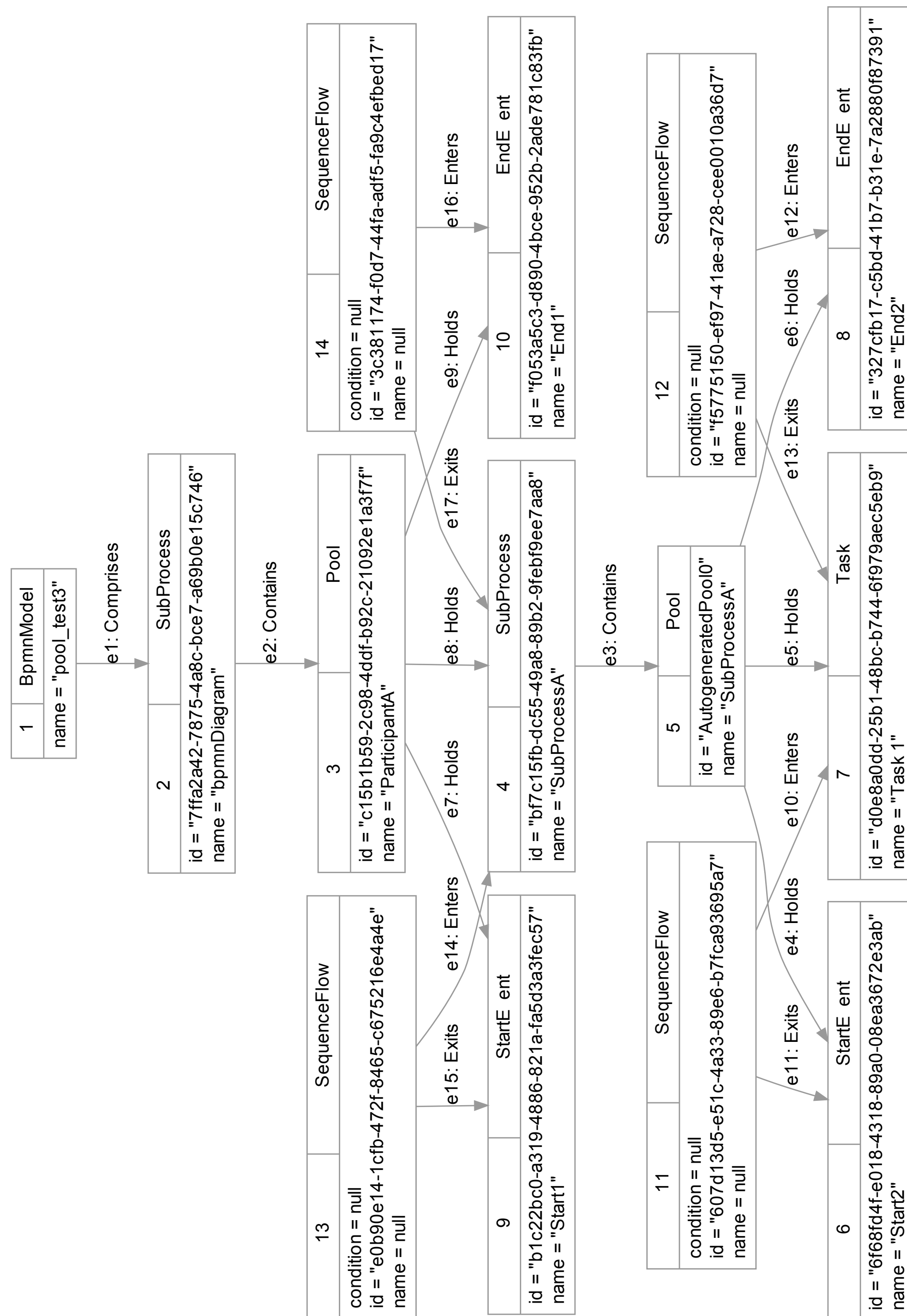


Abbildung 12.18.: Bei der Überführung in ein *BPMN*-Metamodell-konformes Modell tritt der nicht-graphisch dargestellte *BPMN* ***Pool*** im *BPMN* ***Sub-Process*** *SubProcessA* in Erscheinung. Die entsprechende *BPMN*.***Pool***-Instanz *v5* teilt sich den Namen mit der *BPMN*.***SubProcess***-Instanz in der sie enthalten ist – *v4* – und erhält eine bei der Konvertierung automatisch generierte *id*.

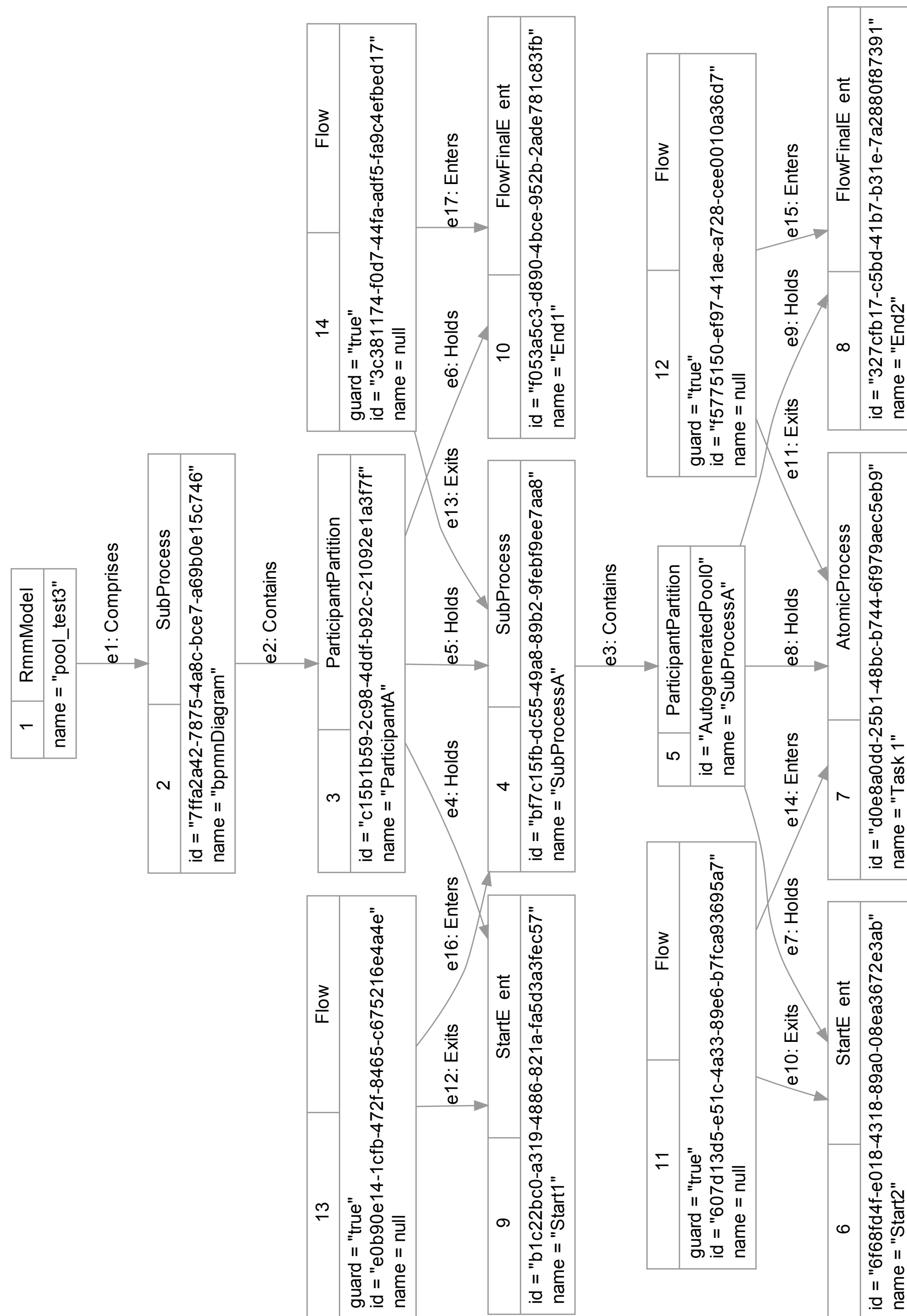


Abbildung 12.19.: Bei der Überführung in ein Referenzmetamodell-konformes Modell wird die *BPMN*.***Pool***-Instanz aus der *BPMN*.***SubProcess***-Instanz in eine RMM.***ParticipantPartition***-Instanz überführt – Knoteninstanz *v5*.

12.5. Beispiel: Geschäftsprozess *sellTrainticket*

Die Darstellung des, als Einführungsbeispiels für die *UML* Aktivitätsdiagramme, *BPMN* und EPK verwendeten, Geschäftsprozesses *sellTrainticket* als *BizAgi* Diagramm, unterscheidet sich nur geringfügig von der Darstellung des *BPMN*-spezifikationskonformen Diagrammes (siehe Abb. 12.20). Der einzige Unterschied besteht in der Notation des einzigen *BPMN* ***Data Objects*** *:Ticket*, welcher im *BizAgi* Diagramm im Bezeichner des transportiereden *BPMN* ***Sequence Flows*** notiert werden muss (siehe Abb. 12.21).

Die Transformation dieses Geschäftsprozessmodells in ein *BPMN*-Metamodell-konformes (siehe Abb. 12.22) bzw. Referenzmetamodell-konformes Modell (siehe Abb. 12.23), kann problemlos anhand der in dieser Arbeit erstellten Konverter durchgeführt werden.

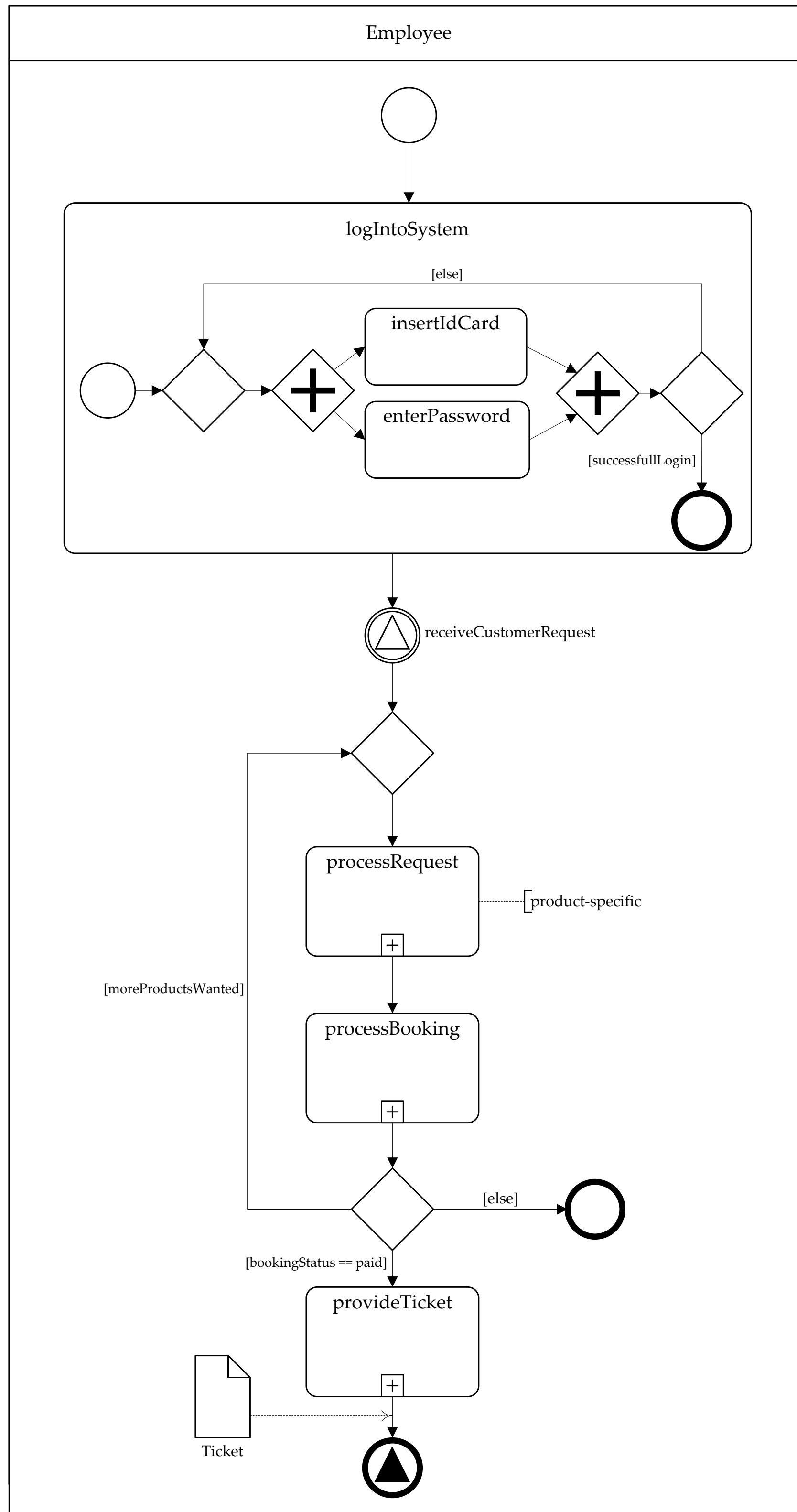


Abbildung 12.20.: *BPMN*-spezifikationskonforme Notation des Geschäftsprozesses *sellTrainticket*.

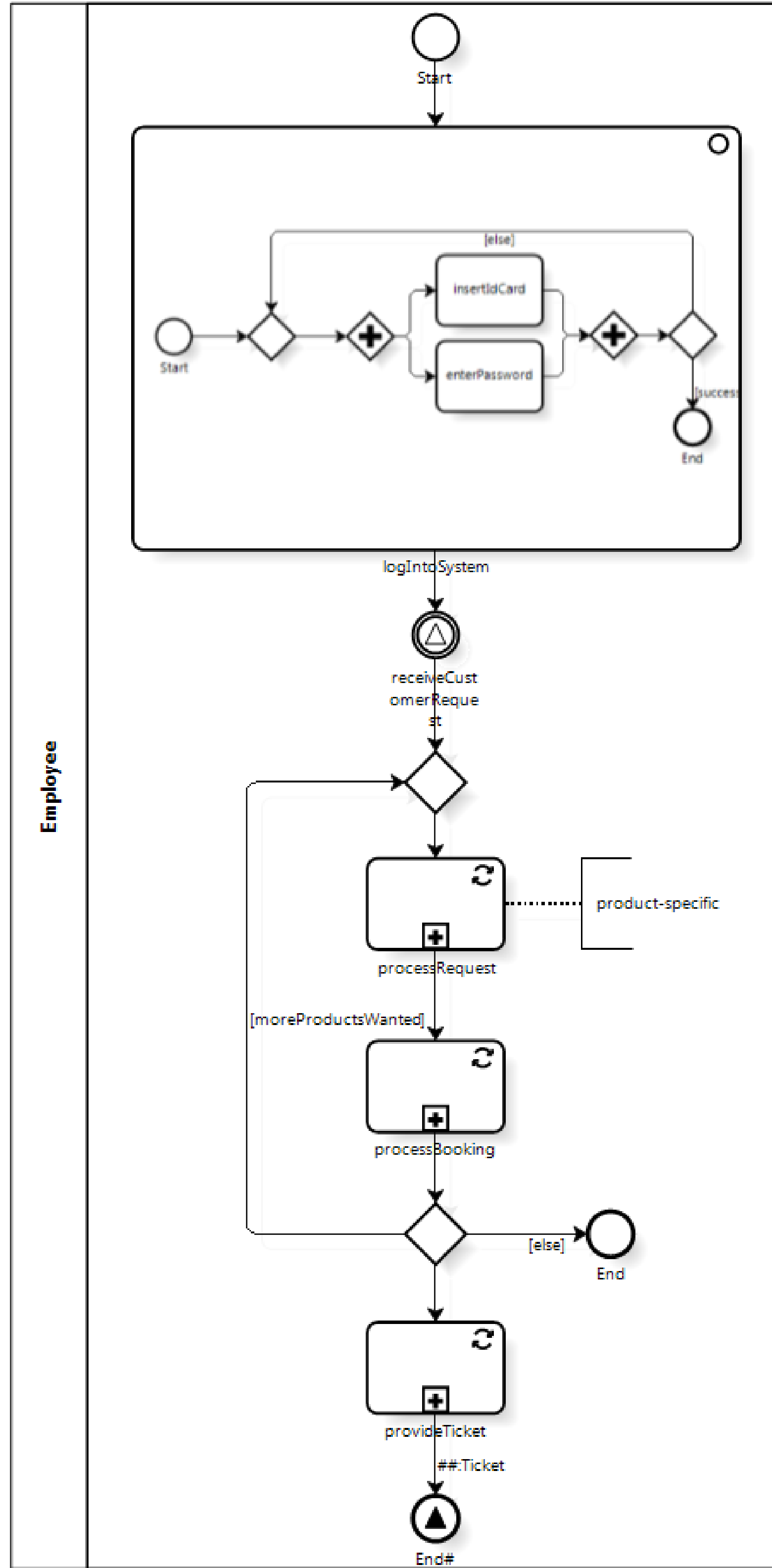


Abbildung 12.21.: *BizAgi* Diagramm des Geschäftsprozesses *sellTrainticket*.

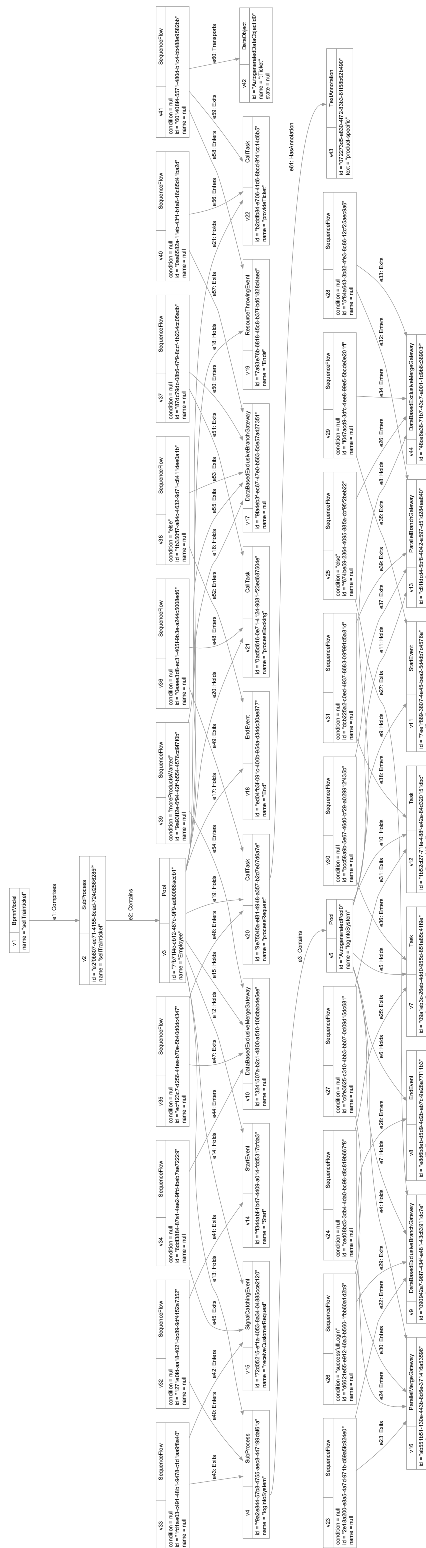


Abbildung 12.22.: *BPMN*-Metamodell-konforme Darstellung des Geschäftsprozessmodells *sellTrainticket*.

Abbildung 12.23.: Referenzmetamodell-konforme Darstellung des Geschäftsprozessmodells *sellTrainticket*.

Teil V.

Fazit

13. Zusammenfassung

Für eine modell-basierte Migration steht die Erhebung von Modellen an erster Stelle. Für die Überführung von Altsystemen in Neusysteme werden daher im *SOAMIG*-Projekt, neben Code- und Architekturmodellen, auch die betrieblichen Abläufe in Form von Geschäftsprozessmodellen erhoben und verknüpft sowie für die Migration in *JGraLab* vorbereitet. Für die Erhebung dieser Modelle im Allgemeinen, und die Erhebung von Geschäftsprozessmodellen im Besonderen, gibt es eine Vielzahl verschiedener Notationen, welche in unterschiedlichen Unternehmen und Geschäftsfeldern benutzt werden.

Um die Weiterverarbeitung von Geschäftsprozessmodellen, welche in unterschiedlichen Notationen erhoben wurden, in *JGraLab* zu vereinfachen, stand die Erstellung einer Referenz für Geschäftsprozessmodellierungssprachen im Vordergrund dieser Arbeit. Hierfür wurden zuerst die Sprachkonzepte und Notationselemente aus drei der verbreitesten Notationen – den *UML* Aktivitätsdiagrammen, der *BPMN* und der EPK – bestimmt und beschrieben, welche für die Modellierung von Geschäftsprozessen im Rahmen des *SOAMIG*-Projekts benötigt werden. Hierbei wurde bereits auf eine möglichst hohe Kompatibilität geachtet und auf Notationselemente oder Sprachkonzepte verzichtet, welche schwer in den jeweils anderen Notationen realisierbar wären.

Auf Gemeinsamkeiten zwischen Sprachkonzepten und Notationselementen der verschiedenen Notationen aufbauend, wurden Referenzkonzepte und Referenzelemente beschrieben. Für Inkompatibilitäten zwischen den Notationen wurden Lösungen ersonnen und beschrieben, wodurch die reibungslose Abbildung von Geschäftsprozessmodellen aus einer Notation in die jeweils anderen Modellierungssprachen und in die Referenz ermöglicht werden.

Anhand der gewonnenen Erkenntnisse zu den Notationssprachen sowie den benötigten Referenzkonzepten und Referenzelementen wurden passende Metamodelle erhoben und beschrieben. Weiterhin wurden, exemplarisch für die Modellierungssprache *BPMN* und das Modellierungswerkzeug *BizAgi Process Modeler*, Konverter erstellt und vorgestellt. Diese Konverter erlauben die Transformation von Modellen welche mit *BizAgi*, unter Benutzung der in dieser Arbeit vorgestellten Sprachkonzepte und Notationselemente, erstellt wurden in ein *BPMN*-Metamodell-konformes Modell, und schließlich die Überführung dieses Modells in ein Referenzmetamodell-konformes Modell durch Anwendung der vorgestellten Transformationsregeln.

Die in dieser Arbeit erstellten Metamodelle, für die dieser Arbeit zugrunde liegenden drei Geschäftsprozessmodellierungssprachen, und Referenzmetamodelle in Kombination mit Transformationsregeln und Konvertern, welche sich an den exemplarisch erstellten Regeln und Konverter orientieren, ermöglichen eine sprach- und tool-unahängige Erhebung von Geschäftsprozessmodellen im Kontext des *SOAMIG*-Projekts und eine einhergehende, leichtere Weiterverarbeitung dieser Modelle in *JGralab*.

Teil VI.

Appendix

Appendix A – *Compact Disc*-Inhalt

Die beiligende *Compact Disc* enthält folgenden Inhalt:

- die Ausarbeitung zu dieser Arbeit als *PDF-A*-Dokument,
- die Modelle und Diagramme aus dem Validierungskapitel als *PDF*-Dokumente,
- die im Rahmen dieser Arbeit erstellten *UML-*, *BPMN* und EPK-Metamodelle sowie das Referenzmetamodell zusammengefasst unter einem *IBM Rational Software Architect* Projekt,
- *Eclipse*[1]-Projekte für jede der drei Sprachmetamodelle und das Referenzmetamodell, welche anhand des darin enthaltenen *Build-files* mit *Apache Ant*[2] (getestet mit der aktuellen Version vom 8. Februar 2010) und *JGraLab* in *Java*-Interfaces und -Klassen überführt werden können,
- die im Rahmen dieser Arbeit erstellten Konverter als *Java Jar*-Datei,
- die kommentierten Quelltexte der Konverter in einem *Eclipse*-Projekt,
- die aktuelle *JGraLab*-Version[3] als *Java Jar*-Datei, sowie
- die *Windows*-Version der Software *BizAgi Process Modeler*[4] mit der die Beispielmodelle erstellt und auf dessen Basis die Konverter geschrieben wurden.

[1] `http://www.eclipse.org/`
[2] `http://ant.apache.org/`, getestet mit *Apache Ant* Version 1.8.0 vom 8. Februar 2010
[3] `http://www.uni-koblenz.de/~ist/JGraLab`, vom 25. März 2009
[4] `http://www.bizagi.com/`, Version 1.4.2

Literaturverzeichnis

[All08] ALLWEYER, Thomas: *BPMN - Business Process Modeling Notation: Einführung in den Standard für die Geschäftsprozessmodellierung*. Books on Demand GmbH, Norderstedt, 2008

[GK92] G. KELLER, A.-W. S. M. Nüttgens N. M. Nüttgens: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. (1992)

[Jac06] JACOBSON, Ivar: *Ivar Jacobson on UML, MDA, and the future of methodologies*. `http://www.infoq.com/interviews/Ivar_Jacobson`. Version: 10 2006, Abruf: 21.09.2009

[JZ07] J. ZIEMANN, A. W.: *Model-based Migration to Service-oriented Architectures*. `http://www.uni-koblenz.de/~ist/documents/Winter2007MMT.pdf`. Version: 2007, Abruf: 17.02.2010

[Kec06] KECHER, Christoph: *UML 2.0: Das umfassende Handbuch*. 2. Auflage. Galileo Press, Bonn, 2006

[Mey97] MEYER, Bertrand: *UML: The positive Spin*. `http://archive.eiffel.com/doc/manuals/technology/bmarticles/uml/page.html`. Version: 1997, Abruf: 21.09.2009

[Mie02] MIELKE, Dr. C.: *Geschäftsprozesse: UML-Modellierung und Anwendungs-Generierung*. Spektrum Akademischer Verlag GmbH, Heidelberg, 2002

[NR02] NÜTTGENS, M. ; RUMP, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). (2002), S. 14

[OMG09a] OMG: *Business Process Modeling Notation*. `http://www.omg.org/docs/formal/09-01-03.pdf`. Version: Januar 2009

[OMG09b] OMG: *OMG UML, Superstructure*. `http://doc.omg.org/formal/2009-02-02.pdf`. Version: Februar 2009

[SOA09] *SOAMIG - Projektbeschreibung*. `http://www.uni-koblenz-landau.de/koblenz/fb4/institute/IST/AGEbert/projekte/soamig/?set_language=de`. Version: 06 2009, Abruf: 21.06.2009

[ST05] SCHEER, A.-W. ; THOMAS, O.: Geschäftsprozessmodellierung mit der Ereignisgesteuerten Prozesskette. In: *Das Wirtschaftsstudium* 34 (2005), 8-9 (2005), S. 1069–1078

[Sta06] STAUD, Professor Dr. J.: *Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für Betriebswirtschaftliche Standardsoftware*. 3. Auflage. Springer-Verlag, Berlin Heidelberg, 2006