Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Situationsabhängige Modellierung und Ausführung von Choreographien

Domas Mikalkinas

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Dr. h.c. Frank Leymann
Betreuer/in:	Kálmán Képes, M.Sc.
Beginn am:	6. September 2019
Beendet am:	6. März 2020

Kurzfassung

Die Modellierung von komplexen vernetzten Systemen ist eine schwierige Aufgabe. Sie erfordert die Einbindung vieler Interaktionspartner und heterogener Systeme. Jedoch ermöglicht sie auch die Erschaffung situationsadaptiver Systeme, die auf Situationen und Ereignisse der echten Welt reagieren und passende Aktionen ausführen können. Die technologische Entwicklung produziert dabei immer komplexere Systeme, was bei der Modellierung dieser zu einem erhöhten Zeitaufwand und größerer Fehleranfälligkeit führt. Zwar können mithilfe von Workflow-Sprachen wie der Business Process Model and Notation (BPMN) diese Probleme teilweise abgemindert werden, allerdings nicht in einem zufriedenstellenden Rahmen. Diese Thesis schlägt deswegen einen Ansatz zur situationsabhängigen Modellierung von Choreographien vor. Der Ansatz umfasst die Erstellung eines Modellierungskonzepts, welches Situationen als eigenständig modellierbare Elemente einführt, die mit Choreographiediagrammen kombiniert und mittels Transformation in ein Standard-Kollaborationsdiagramm umgewandelt werden. Dieses Kollaborationsdiagramm stellt ein Prozessskelett dar, welches die Interaktionen zwischen Teilnehmern abbildet und durch Anreicherung von Aktivitäten durch einen Modellierer zu einem ausführbaren Prozessmodell weiterentwickelt werden kann. Das Konzept der situationsabhängigen Modellierung wird prototypisch im BPMN-Editor “Camunda Modeler“ implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

Akronyme

BPEL Business Process Execution Language. 40

BPM Business Process Management. 67

BPMN Business Process Model and Notation. 3

CSS Cascading Style Sheets. 68

HTML Hypertext Markup Language. 68

IoT Internet of Things. 17

IT Informationstechnologie. 17

JSON JavaScript Object Notation. 11

OMG Object Management Group. 24

SitME Situation-Aware Workflow Modelling Extension. 39

WfMC Workflow Management Coalition. 23

WfMS Workflow Management System. 31

1 Einleitung

Die heutige technologische Entwicklung führt zu größerer Vernetzung von Systemen. Dabei erzeugt eine Vielzahl unterschiedlicher neuer Geräte innovative Möglichkeiten zur Messung und Verarbeitung von Daten und ermöglicht neuartige Dienste. Diese Entwicklung ist rasant und täglich werden neue Systeme und Geräte mit Netzen verbunden oder neue Netze geschaffen. Das Paradigma, welches verschiedenste Geräte mit herkömmlichen Informationstechnologie (IT)-Diensten über das Internet verbindet, wird auch Internet of Things (IoT) genannt. IoT wird heutzutage von großen Firmen genutzt [YAH+17]. IoT ist allerdings nicht nur relevant für Entwickler und Unternehmen, sondern insbesondere auch für Endbenutzer. Unter dem Oberbegriff “Smart Home“ bringen Produkte wie “Amazon Alexa“ oder “Google Assistant“ diese Entwicklung in die Wohnzimmer der Bevölkerung. Es gibt smarte Glühbirnen die per Sprache gesteuert werden, vernetzte Lautsprecher für eine einheitliche Musikerfahrung im ganzen Haus, sowie WLAN-Steckdosen, mit welchen man auch Geräte, die nicht für IoT entwickelt wurden, in das vernetzte System einbinden kann.

Neuartige Sensoren erlauben eine bessere Abbildung der Welt, Fortschritte in der Robotik ermöglichen neue Wege, um die Welt zu manipulieren. Die Kombination der neuen Hardware ermöglicht neue Anwendungsgebiete. Entsprechend können neue Systeme viel besser eigenständig handeln und sich an Umstände anpassen. Solche adaptiven Systeme haben das Potenzial viel Arbeit, die zurzeit noch durch menschliche Intervention erledigt werden muss, zu erledigen, aber auch durch die Automatisierung neue Standards in Sicherheit und Qualität von Prozessen zu ermöglichen.

Diese Entwicklung bringt jedoch nicht nur neue Innovationen, sondern führt auch zu Problemen. Zum einen gibt es eine Vielzahl von Anbietern, was zu einer großen Menge an unterschiedlichen Schnittstellen führt. Zum anderen führt es zu sehr komplexen Systemen. Der Aufwand diese Systeme zu implementieren ist groß, viele verschiedene Technologien und Parameter müssen integriert werden. Insbesondere bei Systemen die dynamisch auf bestimmte Situationen oder Ereignisse reagieren sollen, führt die Notwendigkeit für alternative Systemprozessabläufe zu einer exponentiell wachsenden Zahl von Prozessabläufen und entsprechendem Programmieraufwand.

Workflow-basierte Sprachen können verwendet werden, um diese Komplexität zu beherrschen und um die Kommunikation zwischen den Komponenten eines Systems zu definieren. Außerdem sind solche Sprachen häufig abstrakt definiert und ermöglichen so die Abläufe zunächst genau und geräteunabhängig zu definieren und das Modell erst danach mit den, für die heterogenen Geräte notwendigen, Informationen anzureichern. Der De-facto Standard unter den Workflow-basierten Sprachen ist BPMN. BPMN bietet eine grafische Notation, um Geschäftsprozesse zu modellieren. Dabei gibt es unterschiedliche Submodelle für verschiedene Anwendungsbereiche. So lassen sich interne Abläufe in Orchestrationsmodellen festhalten oder aber mit Choreographiemodellen die Interaktion zwischen Teilnehmern eines Systems, welches sich für die Interaktion zwischen verschiedenen IoT-Geräten eignet [MD17].

Choreographiemodelle bieten jedoch nur eine bestimmte Sichtweise auf ein System und müssen aufwändig in, für die Teilnehmer ausführbare, Prozessmodelle transformiert werden. Ebenso bieten sie keine Lösung, um den exponentiellen Implementierungsaufwand bei komplexen Systemen einzuschränken; wenn man sich beispielsweise ein gutes Smart Home-System vorstellt, dann muss dieses auf eine Vielzahl an Situationen und Ereignissen reagieren können. Mögliche Situationen und Ereignisse könnten etwa der unerwartete Ausfall von Geräten, das Fehlen von Ressourcen oder auch Befehle durch den Nutzer sein. All diese Möglichkeiten bilden im Prozessmodell einen eigenen Pfad ab und jeder mögliche Ausführungspfad muss einzeln modelliert und die Schnittstellen definiert werden [BBK+13]. Auch wenn man solch ein Modell erweitert, beispielsweise wenn man zusätzliche Situationen oder Ereignisse abbilden möchte, kann dies zu Problemen führen, da man unter Umständen direkt in die bereits modellierten Prozesse eingreifen und diese verändern muss. Aufgrund der hohen Komplexität und Unübersichtlichkeit ist dabei die Gefahr ungewollte Nebeneffekte in der Prozesslogik einzuführen sehr groß.

1.1 Motivation

Adaptive Systeme sind entsprechend sehr komplex und schwierig zu implementieren. Sie müssen eine Vielzahl von Situationen abbilden, sowie definieren, wie diese Situationen behandelt werden müssen. In menschlicher Sprache ist es einfach adaptive Situationen zu beschreiben, da es einer dem Menschen intuitiven Darstellung der Welt entspricht. Man weiß was mögliche Situationen sind und kann durch simple “Wenn-Dann“-Bedingungen Lösungen beschreiben. In interagierenden Softwaresystemen können Situationen jedoch nicht so intuitiv beschrieben werden. Wenn man beispielsweise BPMN betrachtet, dann liegt der Fokus bei der Modellierung auf Tasks und ihren Kontrollflüssen. Die einzelnen Elemente einer Situation werden über mehrere Teilnehmer verstreut und teilweise auch durch andere Tasks unterbrochen, außer die Situation umfasst zufällig nur eine einzige Komponente. Betrachtet man etwa Abbildung 1.1, dann erkennt man ein Kollaborationsdiagramm in BPMN. Dieses Diagramm stellt einen Prozess dar, in dem ein Nutzer einer Kaffeemaschine eine Nachricht und damit einen “Kaffee-Kochen“-Prozess initiiert. Nach erfolgreichem Abschluss schickt die Kaffeemaschine dem Nutzer die Bestätigung über den erfolgreichen Abschluss des Prozesses. Falls die Kaffeemaschine jedoch keinen Kaffee aufkochen kann, sendet sie dem Nutzer eine Nachricht über den Fehlschlag des Prozesses. Diese beiden Pfade stellen unterschiedliche Situationen dar, werden aber nicht explizit im Diagramm modelliert. Wenn man nun eine diese Situation abändern oder eine neue Situation hinzufügen wollen würde, ist es nicht unwahrscheinlich, dass man in beiden Prozessen Änderungen vornehmen müsste. Dies würde zum einen die Übersichtlichkeit senken, zum anderen würde man Gefahr laufen unerwünschte Nebeneffekte einzufügen. In einem kleinen und übersichtlichen Beispiel, wie dem hier vorgestellten, wäre das noch umsetzbar, in einem großen und weit vernetzten System wäre dies jedoch mit erheblichem Aufwand verbunden. Die Hardware für solche Systeme ist also vorhanden, allerdings mangelt es an guten Modellierungsmöglichkeiten, mit denen man solche Systeme mit akzeptablem Aufwand umsetzen kann.

1.2 Zieldefinition

Es besteht daher die Notwendigkeit ein Modellierungskonzept für die Modellierung von Situationen zu entwickeln. Der Definition von Greenberg [Gre01] folgend, soll das Konzept die Modellierung diese Schritte unterstützen:

- Die einzelnen Situationen definieren
- Die Elemente, die diese Situationen akkurat darstellen, definieren
- Die Aktionen, welche beim Eintreten der Situation ausgeführt werden sollen, definieren

Außerdem sollte das Konzept in einen bereits existierenden Standard transformierbar sein, damit der Weg von Konzept zu Verwendung möglichst gering ist. Hier bietet sich BPMN an, da es den De-facto Standard für Workflow-Modellierung darstellt.

Neben der Wahl der Modellierungssprache muss auch die Diagrammart, in welche das Modell transformiert werden soll, definiert werden. Insbesondere muss festgelegt werden, ob das Zieldiagramm eine Orchestration oder eine Choreographie darstellen soll. Betrachtet man IoT, dann erkennt man, dass die heutigen IoT-Architekturen häufig auf zentralisierten Ansätzen basieren [MSPZ18]. Jedoch gibt es auch immer häufiger Stimmen, welche dezentrale Architekturen fordern. Die Gründe hierfür sind unter anderem fehlende Effizienz von zentralen Systemen, Bedenken bezüglich der Privatsphäre, sowie die wachsende Menge von Daten, welche zunehmend die Netzwerke überfordern [HHBS18; MSPZ18; PCS18; SNPB14]. Konzepte wie “Edge Computing“ und “Blockchain“ stellen dabei mögliche Lösungen für diese Probleme dar [HHBS18; PCS18]. Außerdem existieren viele Ansätze, um Choreographien in Orchestrationen umzuwandeln und nur wenige, welche den umgekehrten Weg gehen [Kha10; MH08]. Aus diesen Gründen wird in dieser Thesis die Transformation in ein Standard-BPMN-Kollaborationsdiagramm umgesetzt.

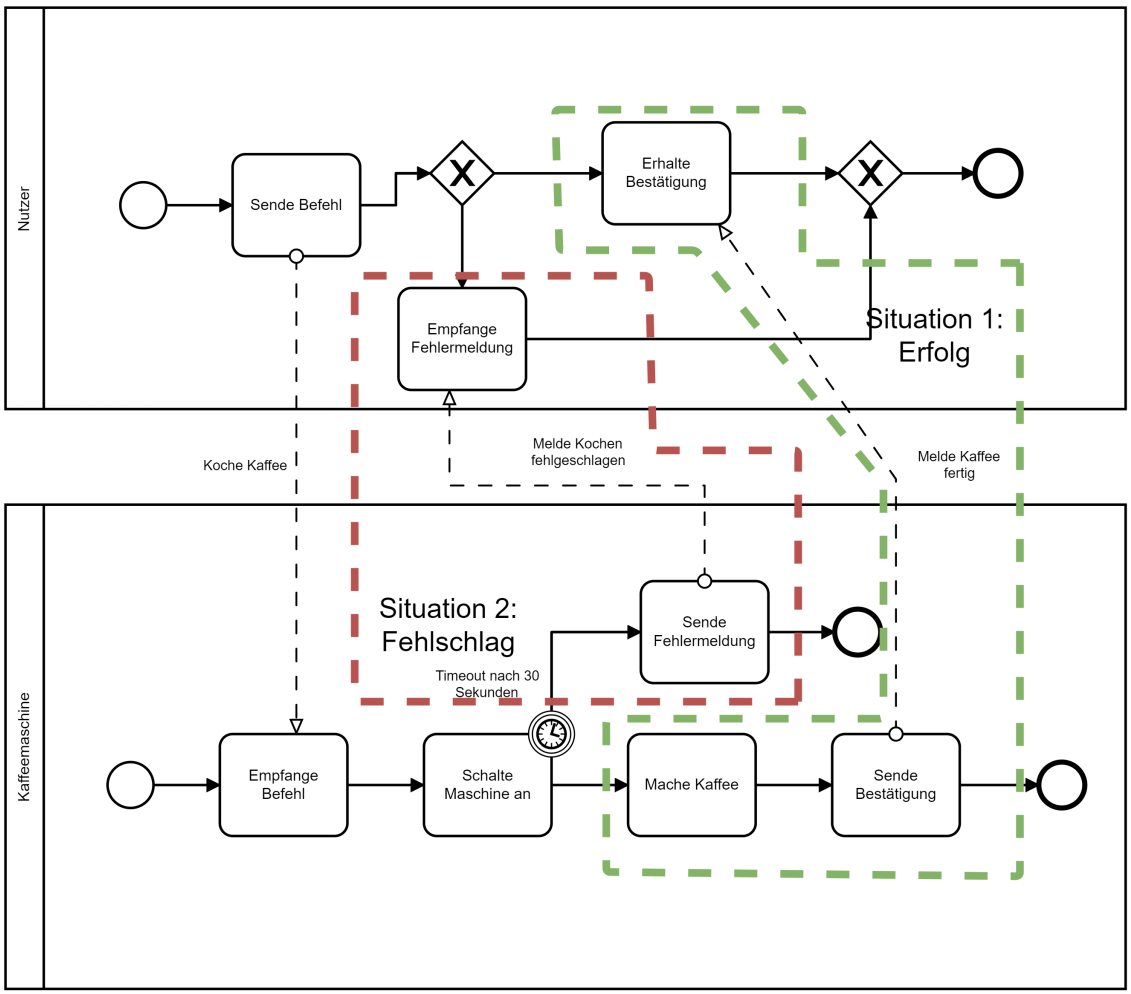


Abbildung 1.1: Darstellung von Situationen in einem Kollaborationsdiagramm. Die Situationen im Diagramm sind mit gestrichelten Kästen umrandet

Diese Thesis erforscht die Möglichkeiten einer solchen Modellierungsmethode und beschreibt die Implementierung eines Prototyps mit den Ergebnissen der Forschung. Dabei baut das Modellierungskonzept auf den Choreographiemodellen von BPMN auf und erweitert diese um Elemente, welche es ermöglichen situationsabhängige Modelle zu modellieren, die erweiterbar sind. Weiterhin wird ein Transformationsalgorithmus implementiert, welcher die Choreographiemodelle in Kollaborationsmodelle umwandelt. Der Prototyp ist im Camunda Modeler implementiert, welcher das BPMN.io-Framework umsetzt. Da der Camunda Modeler nicht nativ Choreographiemodellierung unterstützt, wird zusätzlich die chor-js-Erweiterung des Hasso-Plattner-Instituts genutzt, welche eine Bibliothek für Choreographiemodellierung auf BPMN.io-Basis zur Verfügung stellt.

1.3 Gliederung der Thesis

Im Weiteren beschreibt die Thesis im Kapitel 2 zunächst die Grundlagen, die notwendig sind, um die Modellierungsmethode zu verstehen. In Kapitel 3 werden Forschungsarbeiten vorgestellt, welche einen Einfluss auf die Entwicklung des Konzepts hatten. In Kapitel 4 wird ein Use Case vorgestellt, anhand welchem das Modellierungskonzept des Ansatzes erklärt wird. Das Konzept selbst wird in Kapitel 5 ausführlich beleuchtet, während die Implementierung in Kapitel 6 behandelt wird. Das Konzept wird in Kapitel 7 zusammenfassend diskutiert. Schließlich wird die Thesis in Kapitel 8 nochmals zusammengefasst und ein Ausblick auf mögliche zukünftige Forschung gegeben.

2 Grundlagen

Dieses Kapitel erklärt die Grundlagen, welche nötig sind, um den, in dieser Thesis entwickelten, Ansatz zu verstehen. Insbesondere BPMN wird diskutiert, da dies die Modellierungssprache ist, die dem Konzept dieser Thesis zugrunde liegt. Die anderen Grundlagen, wie etwa Exceptions, werden dabei, wenn nötig, im Kontext von BPMN betrachtet.

2.1 Workflows

Die Workflow Management Coalition (WfMC) beschreibt *Workflows* wie folgt: “Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal“ [HH95]. Die bewusste Verwendung des Begriffs “business goal“, also Geschäftsziel, zeigt, dass Workflows eng mit dem Wirtschaftsbereich zusammenhängen, in dessen Kontext das Konzept auch entstanden ist [Wes19]. Um Unternehmensstrukturen zu beschreiben und zu definieren, wurde in den 1990er Jahren zunehmend der Fokus auf *Prozesse* gelegt. Diese Prozesse, auch *Geschäftsprozesse* genannt, werden als eine Sammlung von Aktivitäten beschrieben, welche ein oder mehrere Eingaben erhalten. Unter Verwendung dieser Eingaben werden Ausgaben erzeugt, welche einen Wert für den Kunden haben [HC09]. Gleichzeitig deutet der Begriff “automation“, also Automatisierung, darauf hin, dass das Ziel von Workflows die Einbindung von IT in solche Prozesse und die Überbrückung zwischen Geschäfts- und IT-Aspekten ist.

Eines der grundlegenden Konzepte von Workflows sind *Activities*. In Activities werden die Aufgaben beschrieben, die in einem Geschäftsprozess ausgeführt werden müssen. Activities bestehen aus mehreren *Tasks*. Ein Task ist dabei definiert als eine logische Arbeitseinheit. Tasks sind notwendig, um Workflows strukturieren zu können. Ein Task wird atomar ausgeführt und muss, wenn er im Fehlerfall abgebrochen wird, vollständig von Beginn an wiederholt werden. Die Atomarität eines einzelnen Tasks hängt jedoch vom Kontext ab. Wenn beispielsweise in einem Prozess der Kunde den Kauf eines Buches anstößt und die Kaufanfrage an den Händler versendet, ist für den Kunden nicht sichtbar, welche Schritte der Händler ausführt, um seinen Teil des Prozesses im Workflow auszuführen. Für den Kunden ist lediglich interessant, dass der Kauf erfolgreich abgewickelt wird, entsprechend sieht er alle Handlungen des Händlers als einen Task an. Der Händler teilt seinen Teil des Workflows jedoch in eine Vielzahl von Tasks auf: Er bearbeitet den Eingang der Ware, schreibt eine Rechnung, versendet diese und bucht das Geld ab [AH02; LR00].

Die Tasks in einem Workflow können manuell von Menschen oder automatisiert von Maschinen ausgeführt werden. Falls an einem Task sowohl Mensch als auch Maschine beteiligt sind, handelt es sich um einen *semi-automatischen Task* [AH02].

In einem Geschäftsprozess können mehrere Fälle zusammengefasst werden. Der Prozess definiert die grundsätzlichen Tasks, die für Fälle eines Geschäftsprozesses ausgeführt werden können, kann jedoch auch Änderungen des Ablaufs für bestimmte Fälle definieren. So könnte beispielsweise eine mögliche Änderung beim Kauf einer Ware für den Fall eines bestimmten Buches definiert werden. Der Prozess definiert alle Tasks, die für den Kauf von Waren notwendig sind, besitzt jedoch einen speziellen Task, welcher einen gesonderten Versand für Bücherkäufe definiert. Je nach Fall kann auch die Reihenfolge der Ausführung der Tasks unterschiedlich sein. Des Weiteren können Prozesse wiederverwendet werden und Teil eines anderen Prozesses sein. In dem Fall spricht man von *Subprozessen*. Alle Fälle eines Prozesses haben eine Lebensdauer, deshalb muss der Prozess klar definierte Anfangs- und Endpunkte haben [AH02].

Um die Ausführung bestimmter Fälle in einem Prozess zu ermöglichen, muss das *Routing* beachtet werden. Die Tasks innerhalb eines Prozesses können sequentiell oder, bei einer Aufteilung des Prozesspfads, parallel ausgeführt werden. Es müssen jedoch nicht automatisch alle parallelen Pfade ausgeführt werden, sondern es können auch eine Untermenge von Pfaden zur Ausführung gewählt werden. Ebenso kann es vorkommen, dass ein Task mehr als ein Mal innerhalb eines Falles ausgeführt wird, beispielsweise in einer Schleife [AH02].

Abschließend kann man noch den *Trigger* für die Ausführung von Activities definieren. Zum einen kann eine *Resource*, welche für die Ausführung der Activity definiert ist, die Ausführung initiieren. Im Buchkaufbeispiel wäre das etwa ein Mitarbeiter, welcher anfängt das Buch für den Versand zu verpacken, sobald er das Buch aus dem Lager geliefert bekommen hat. Ebenso könnte der Trigger ein externes Ereignis, etwa der Eingang der Buchkaufanfrage als E-Mail, sein. Auch ein zeitlicher Trigger, wie etwa ein Timer, ist möglich. Wenn beispielsweise die Zahlung für das Buch nach einigen Tagen nicht eingegangen ist, könnte ein Timer die Stornierung der Bestellung auslösen [AH02].

Diese Konzepte stellen die Grundlagen für mehrere Modellierungssprachen. Eine der wichtigsten ist die BPMN, welche für diese Thesis verwendet wurde und im Folgenden genauer erläutert wird.

2.2 BPMN

Die BPMN wurde entwickelt, um Geschäftsprozesse zu modellieren und baut auf den Erfahrungen existierender Praktiken auf. Sie bildet den De-Facto Standard für die Geschäftsprozessmodellierung. BPMN erfüllt das Ziel von Workflows und kann mehrere Abstraktionsebenen abbilden, sodass alle am Geschäftsprozess Beteiligten das Modell verstehen können. So können etwa Personen mit unternehmerischem Fokus die Geschäftsprozesse abstrakt modellieren, die technischen Entwickler diese implementieren und, bei der Anwendung, die Geschäftsleute diese Prozesse betreuen und überwachen. Die BPMN wird von der Object Management Group (OMG) betreut [OMG13].

BPMN hat mehrere Diagrammtypen. Mittels BPMN können *Orchestrationen*, also Prozesse innerhalb eines Systems, *Kollaborationen*, d. h. innere Prozesse von Systemen und die Darstellung wie Systeme miteinander interagieren oder *Choreographien*, welche einen Fokus auf die Interaktion zwischen den Systemen legen, abbilden. Zu diesem Zweck unterstützt BPMN mehrere Elemente und Konzepte, deren Terminologie sehr ähnlich zu der von Workflows ist.

Die Elemente sind in vier Kategorien unterteilt und in Abbildung 2.1 dargestellt [Wes19]:

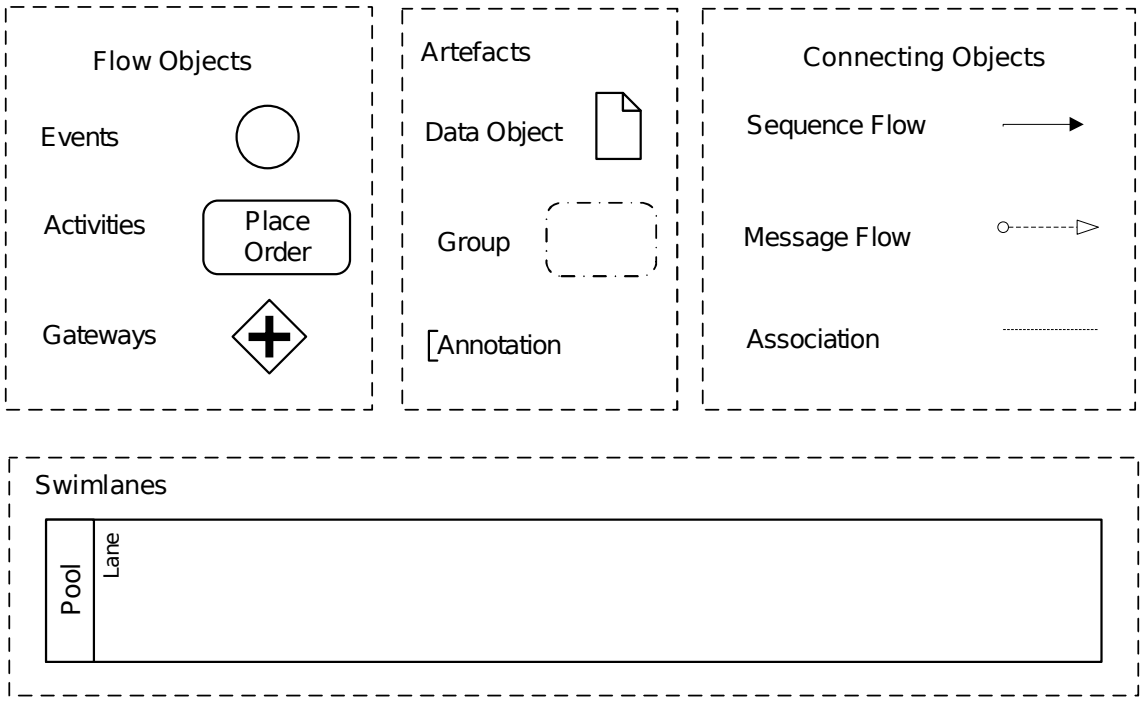


Abbildung 2.1: Elemente, die in BPMN verwendet werden können. Sie sind unterteilt in "Flow Objects", "Artefacts", "Connecting Objects" und " Swimlanes", aus [Wes19]

- *Flow Objects*: Darunter fallen Elemente wie *Events*, *Activities* und *Gateways*. Events bilden Zustände in der echten Welt ab, die relevant für Geschäftsprozesse sind. Events wurden so definiert, dass sie durch die von Workflows bekannten Trigger ausgelöst werden können. Auch Activities folgen der Workflow-Definition: Sie sind Arbeitsschritte, die von Unternehmen in Geschäftsprozessen ausgeführt werden. Prozesse sind strukturierte Abfolgen von Arbeitsschritten, um eine Arbeit im Unternehmen auszuführen. Gateways sind relevant für das Routing, da sie Entscheidungen im Kontrollfluss darstellen. Dabei setzen sie die unterschiedlichen Routing-Optionen des Workflow-Konzepts um.
- *Artefacts*: Artefacts werden verwendet, um zusätzliche Informationen über Geschäftsprozesse darzustellen, welche nicht direkt relevant für den Kontrollfluss des Prozesses sind. Sie haben dabei keinen Einfluss auf eben diesen [FR14]. Ein Beispiel für ein Artefact ist die *Annotation*.
- *Connecting Objects*: Es gibt verschiedene Wege, um Verbindungen zwischen den Elementen herzustellen. Für die Strukturierung von normalen Prozessen werden *Sequence Flows* verwendet. Diese stellen einen Kontrollfluss dar. Mittels *Message Flows* können Nachrichteninteraktionen zwischen verschiedenen Prozessen dargestellt werden. Um Artefacts mit sich selbst oder mit anderen Elementen zu verbinden müssen *Associations* genutzt werden.
- *Swimlanes*: In BPMN können auch Organisationsstrukturen abgebildet werden. Diese nennen sich Swimlanes und werden zwischen *Pools* und *Lanes* unterschieden. Pools sind Organisationen, die an der Interaktion zwischen verschiedenen Geschäftsprozessen teilnehmen. Lanes stellen Einheiten innerhalb der Organisation dar. Pro Pool gibt es nur einen Prozess [OMG13].

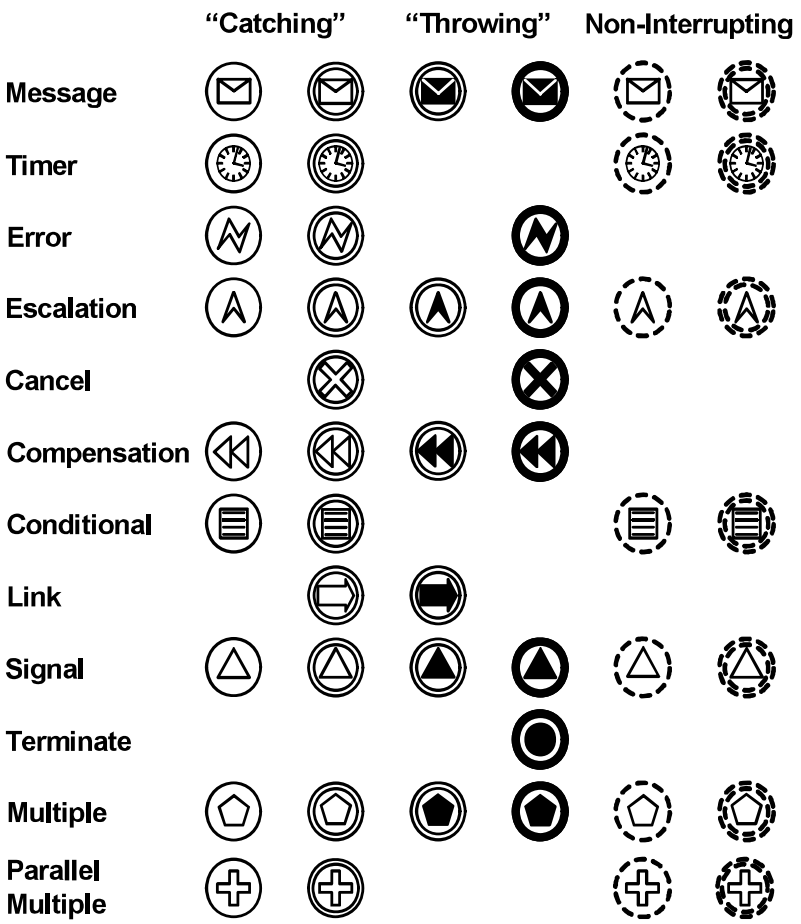


Abbildung 2.2: All verfügbaren Event-Typen aus BPMN, aus [OMG13]

Dem Workflow-Konzept folgend haben Prozesse *Start-* und *End Events*. Außerdem gibt es *Intermediate Events*, welche die Ausführung verzögern oder einen nebenläufigen Ablauf auslösen können. Wie für Workflows vorgesehen, können Events durch äußere Einflüsse, wie Nachrichten, oder innere Einflüsse, wie einen User, ausgeführt werden. Auch zeitlich abhängige Events können modelliert werden. Intermediate Events können eine unterbrechende (verzögernde) oder eine nicht unterbrechende (parallel ablaufende) Wirkung haben. Intermediate Events können an andere Elemente mittels normalen Flows angehängt werden oder aber an die *Boundary* einer Activity. In dem Fall werden ausgehende Sequence Flows als *Exception Flows* angesehen, da der angehängte Ablauf nicht mehr den Standard-Ablauf modelliert, sondern einen Ausnahmepfad darstellt [OMG13]. Eine Übersicht über die Events, die in BPMN verwendet werden, bietet Abbildung 2.2.

2.3 Choreographie

Orchestrationen bilden die Geschäftsprozesse innerhalb einer Organisation oder eines Systems ab. Sie werden zentral gesteuert. Jedoch interagieren die Geschäftsprozesse heutzutage immer stärker mit den Geschäftsprozessen anderer Organisationen. Choreographien werden verwendet, um genau

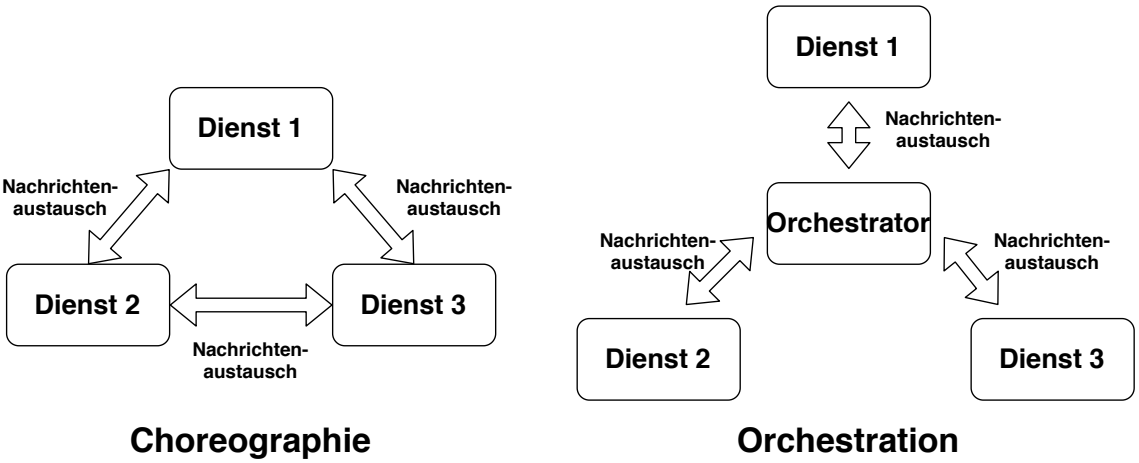


Abbildung 2.3: Unterschied zwischen Orchestration und Choreographie. Orchestrationen werden zentral, Choreographien dezentral gesteuert, nach [RGHH13]

diese Interaktionen darzustellen. Sie stellen eine dezentrale Sicht auf einen Gesamtprozess über Organisationsgrenzen hinweg dar (siehe Abbildung 2.3). Prozesse zwischen verschiedenen Partnern mit eigenen Einflussbereichen benötigen eine besondere Aufmerksamkeit auf die Schnittstellen, da Ambiguitäten zu großen Problemen führen. Wenn der Nachrichtenaustausch nicht genau definiert ist, kann es passieren, dass der Prozess nicht fortgeführt werden kann, da der Nachrichtenempfänger beispielsweise mit einem anderen Format rechnet und die empfangene Nachricht nicht weiterverarbeiten kann. Außerdem ermöglicht man durch klar definierte Schnittstellen den Austausch von Partnerprozessen, da nicht mehr für jeden Partner die Schnittstelle neu definiert werden muss. Hierbei spricht man auch von struktureller Kompatibilität. Die einzelnen Teilnehmer koordinieren sich selbst mittels Nachrichtenfluss [Wes19].

Für die Modellierung von Choreographien werden *Interaction Models* und *Interconnection Models* verwendet [DW11]. In Interconnection Models werden die Kontrollflüsse der jeweiligen Teilnehmer modelliert und die Interaktion zwischen den Teilnehmern mittels Message Flows dargestellt, d. h. die Interaktionen werden nicht als atomare Bestandteile des Modells angesehen. Interconnection Models werden in BPMN als *Kollaborationsdiagramme* modelliert. Interconnection Models können aus verschiedenen Perspektiven betrachtet werden. Die drei verschiedenen möglichen Darstellungsformen zeigt Abbildung 2.4. Die inneren Prozesse eines Teilnehmers können offen gezeigt werden und die Interaktion mit anderen Teilnehmern als Black Box. In dem Fall spricht man von einem *Private model*. Nur die, für die Interaktion relevanten, Informationen können gezeigt werden, dann ist es ein *Public model*. Alternativ kann man auch darauf verzichten den anderen Teilnehmer als Black Box zu modellieren und ebenso seine Prozesse in einem *Collaboration model* darzustellen [BFR19].

In BPMN bestehen Interconnection Models aus Pools, sowie aus den anderen, aus Orchestrationen bekannten, Elementen. Verschiedene Pools innerhalb eines Diagramms stehen für verschiedene Teilnehmer der Interaktion. Die Teilnehmer können nur mittels Message Flows miteinander interagieren.

In Interconnection Models kann es zu Verhaltensinkompatibilität kommen. Die Prozesse der einzelnen Teilnehmer können zwar jeder für sich genommen korrektes Verhalten definieren. Da jedoch unklar ist, in welcher Reihenfolge die Prozesse ausgeführt werden, kann es sein, dass Prozesse an

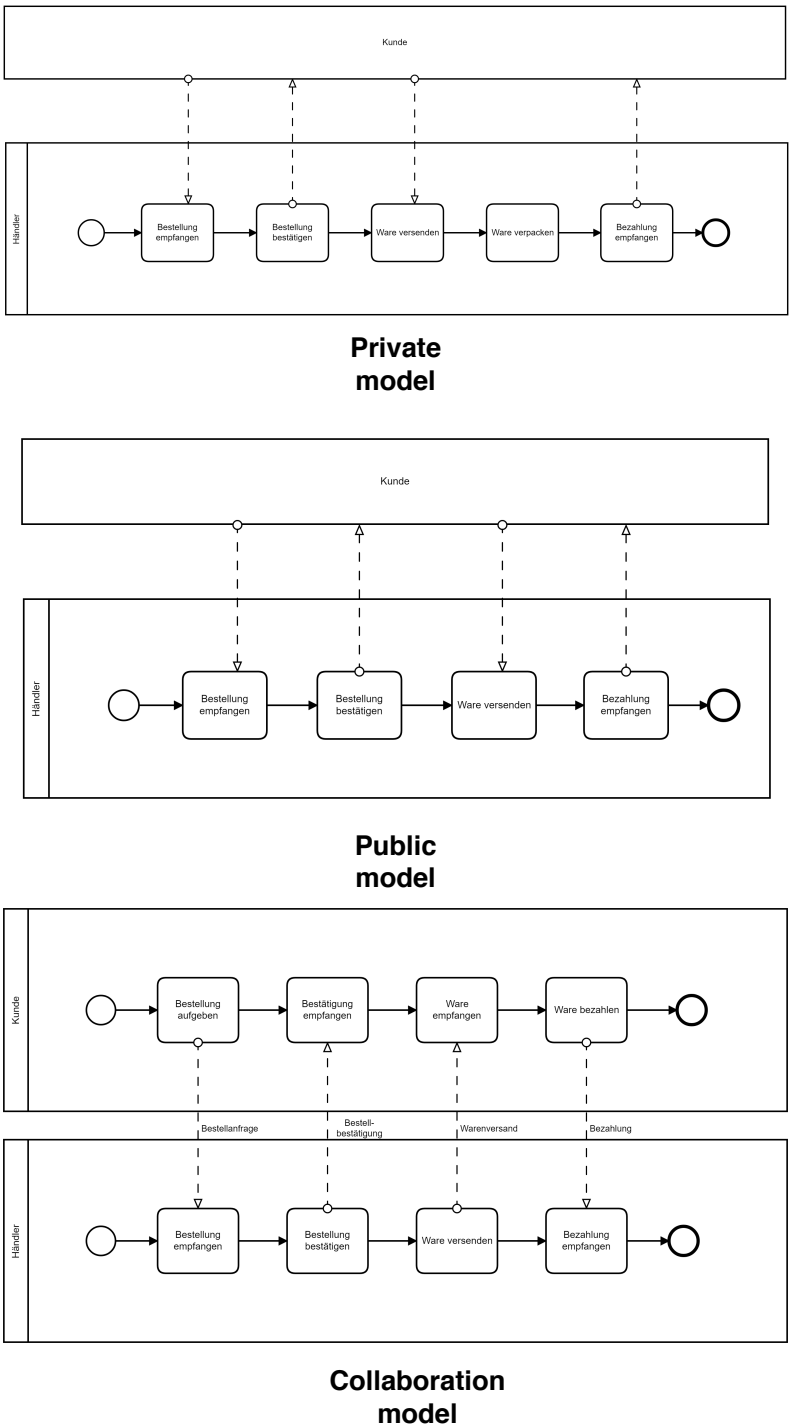


Abbildung 2.4: Die drei Modellformen eines Interconnection Models in BPMN. In BPMN werden sie als Kollaborationsdiagramme dargestellt. Die Modellformen sind “Private model“ (mit internen und Interaktions-Activities eines einzelnen Teilnehmers), “Public model“ (nur Interaktions-Activities eines einzelnen Teilnehmers) und “Collaboration model“ (zeigt auch Activities von anderen Teilnehmern an), nach [BFR19]

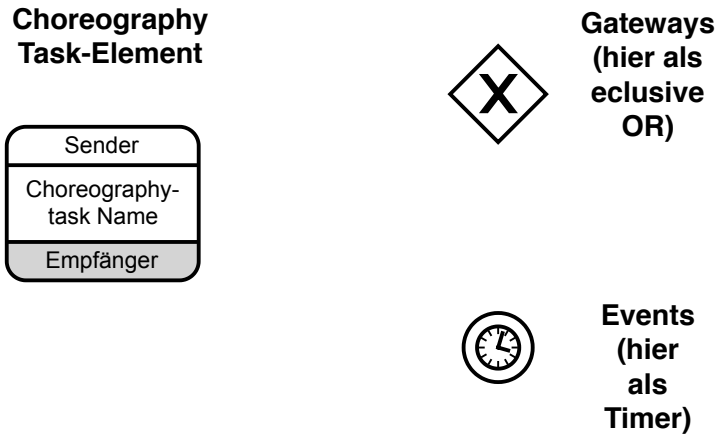


Abbildung 2.5: Die drei Elementtypen, die in Choreographiediagrammen von BPMN unterstützt werden. Dazu gehören Choreography Tasks, welche den Choreographiediagrammen eigen sind, als auch Gateways und Events, die auch in anderen Diagrammarten von BPMN genutzt werden, nach [OMG13]

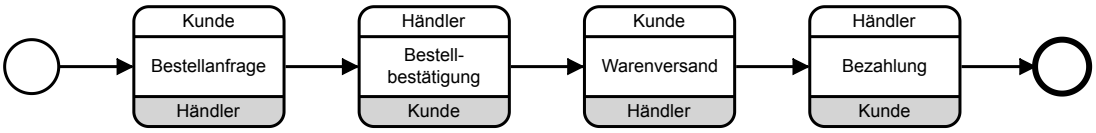
einen Punkt in ihrer Ausführung gelangen, an denen sie auf die Nachricht eines anderen Prozesses warten, während dieser Prozess seinerseits auf eine Nachricht des ersten Prozesses wartet. In dem Fall spricht man von einem "Deadlock". Daher muss die Interoperabilität genau definiert werden, um Konflikte an den Schnittstellen auszuschließen. Solche Deadlocks können mit Interaction Models vermieden werden, da Interaction Models die Verhaltensabhängigkeiten global betrachten. Die richtige Reihenfolge der Interaktion ist demnach automatisch gegeben [BFR19].

In Interaction Models sind Interaktionen die grundlegenden atomaren Elemente. Je nach Modellierungssprache können diese einfachen Nachrichten nach dem "Senden-Empfangen"-Prinzip agieren oder "Anfrage-Antwort"-Nachrichten sein. In BPMN werden Interaction Models in *Choreographiediagrammen* dargestellt. Die Interaktionselemente von Choreographiediagrammen werden *Choreography Tasks* genannt. Neben diesen Choreography Tasks unterstützen Choreographiediagramme auch einige Event-Typen, sowie Gateways [OMG13] (siehe 2.5). Den Unterschied zwischen Choreographiediagramm und Kollaborationsdiagramm in BPMN zeigt Abbildung 2.6.

2.4 Adaption

Häufig werden Begriffe wie "flexibel" und "adaptiv" im Zusammenhang mit IT-Systemen synonym verwendet. Adaption bedeutet die Anpassung eines Prozesses an auftauchende Ereignisse [RW12]. Ereignisse führen beispielsweise dazu, dass der vorgesehene Prozess nicht mehr wie geplant ausgeführt werden kann oder dass die Ausführung dieses Prozesses zu fehlerhaftem Verhalten führt. Der Prozess muss sich also an die neuen Umstände anpassen und besondere Prozesse ausführen, die die neuen Umstände adäquat behandeln. Abweichungen vom Prozess können für bekannte Situationen schon bei der Modellierung beachtet und im Modell definiert werden. Adaption hängt eng zusammen mit dem Begriff *Exception*. Dies wird deutlich, wenn man beispielsweise die Definition von Exception von Adams, welcher sich mit flexiblen Workflows beschäftigt hat, betrachtet: "it [an exception] is simply an event that is considered to be a deviation from the expected control flow or

Choreographie



Kollaboration

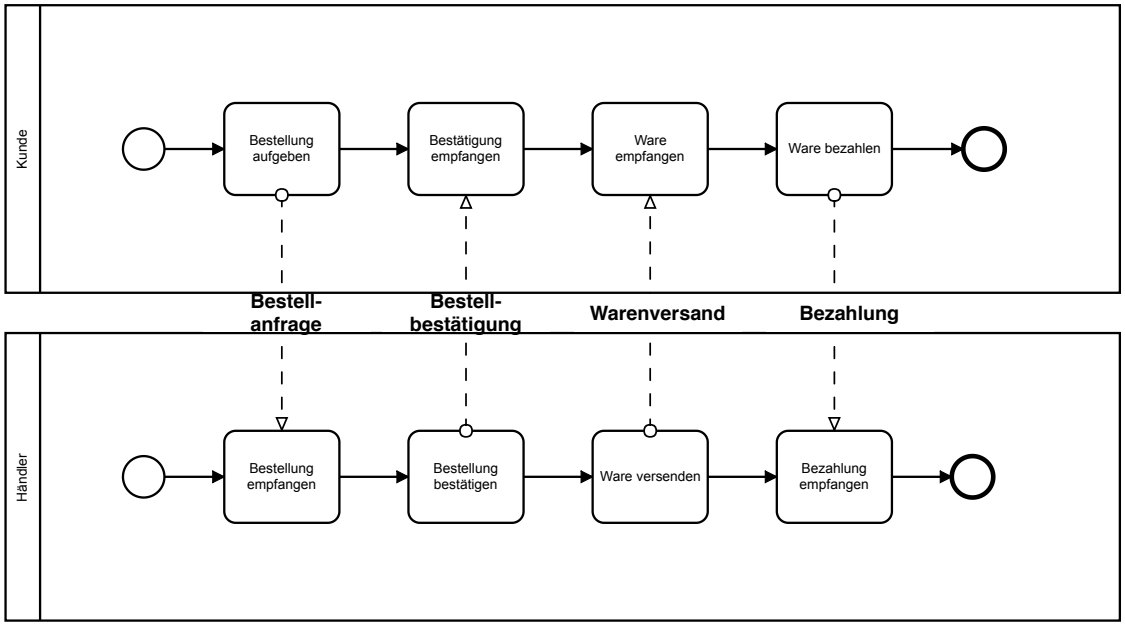


Abbildung 2.6: Unterschied zwischen Choreographie und Kollaboration. Choreographien stellen Interaktionen als Elemente dar, während Kollaborationen Interaktionen als Message Flows zwischen Tasks darstellen

was unaccounted for in the original process model“ [AHEA05]. Ein häufiges Mittel, um mit solchen Exceptions in IT-Systemen umzugehen, ist das *Exception Handling*. Diese Thesis legt den Fokus auf die Behandlung von erwartbaren Ereignissen mithilfe von Exception Handling.

2.5 Exceptions

Unbehandelte Exceptions können zu Fehlern des Systems führen. Für die Definition von Exceptions folgt die Thesis der, im Adaption-Abschnitt vorgestellten, Definition von Adams [AHEA05]. Der Begriff Fehler ist hier jedoch ambivalent und somit unzureichend, da in der englischen Fachliteratur mehrere Begriffe existieren, welche im Deutschen mit Fehler übersetzt werden können. Eine häufige zitierte Definition von Avizienis et al. unterschiedet zwischen *Faults*, *Errors* und *Failures* [ALRL04].

Sie definiert Failure als das beobachtete Verhalten eines Systems, welches von seiner gewollten Funktionsweise abweicht. Ein Error wiederum ist ein fehlerhafter Zustand eines Systems. Die technische Ursache eines Errors hingegen ist ein Fault.

Viele verschiedene Klassifikationen für Exceptions existieren. Eder und Liebhart klassifizieren Workflow Exceptions nach der Art der Behandlung: Exceptions können *Expected* oder *Unexpected* sein oder es können *Basic Failures* oder *Application Failures* sein [EL95]. Basic Failures werden auf dem Systemlevel und Application Failures auf dem Applikationslevel behandelt. Diese Level sind jedoch außerhalb des Umfangs dieser Thesis und werden entsprechend nicht weiter vertieft. Exceptions hingegen werden auf einem höheren Level behandelt. Da Unexpected Exceptions nicht im modellierten Prozess vorgesehen sind, müssen diese bereits bei der Prozessdefinition beachtet werden. Expected Exceptions sollen dagegen vom modellierten Prozess abgefangen werden. Sadiq et al. [SO00] erweitert diese Klassifikation: Expected Exceptions können *Embedded* oder *Separated* sein. Embedded Exceptions werden Teil des Prozess-Workflows. Dies führt zu einer größeren Komplexität des Prozesses. Separated Exceptions werden entweder mittels festgelegter Regeln an das Workflow Management System (WfMS) übergeben, welches den dafür zuständigen Exception Handler aufruft oder es wird ein separater Workflow abhängig von der Fehlerkondition gestartet. Des Weiteren können Exceptions generisch sein oder applikationsspezifisch [HBM08]. So sind beispielsweise eine generische Exception ein Timeout des Timers und eine applikationsspezifische Exception die fehlende Verfügbarkeit eines spezifischen Buches in einem Bestellprozess.

Die meisten Exception Handling-Strategien kümmern sich um Expected Exceptions. Auch in dieser Thesis wird diese Exception-Kategorie behandelt.

2.6 Exception Handling in Workflows

Reichert et al. [RW12] haben Exception Handling in Workflows erforscht und verschiedene Patterns für das Exception Handling vorgestellt. Sie kategorisieren verschiedene Quellen für Exceptions:

- *Activity Failures*: Diese Exceptions entstehen während der Ausführung von Activities. Reichert et al. unterscheiden dabei zwischen *Semantical* und *Technical* Activity Failures. Technical Activity Failures lassen sich auf die verwendeten Technologien zurückführen. So könnte ein Technical Activity Failure ein Fehler im Netzwerk sein, welcher den unerwarteten und abnormalen Abbruch der Activity zur Folge hat. Semantische Activity Failures entstehen dagegen durch unerwartete Situationen. Wenn etwa beim Bestellvorgang eines Buches dieses gerade für den Versand vorbereitet wird und aufgrund einer kurzfristigen Stornierung des Kaufes der Verpackvorgang abgebrochen wird, handelt es sich dabei um einen Semantic Activity Failure. Der Abbruch der Activity ist hierbei nicht abnormal.
- *Deadline Expirations*: Bei einer Deadline Expiration wird einer Activity eine bestimmte Zeit zur Ausführung zugeordnet. Wenn die Ausführung der Activity nicht innerhalb dieses Zeitraums abgeschlossen wird, terminiert die Activity. Dadurch werden Eskalationsprozesse in Gang gesetzt, welche einen alternativen Kontrollpfad definieren.
- *Resource Unavailability*: Diese Exception tritt auf, wenn eine Resource, welche einer Activity zugeordnet worden ist, selbige nicht bearbeiten kann. Entsprechend muss entweder eine andere Resource zugeordnet oder der Vorgang abgebrochen werden.

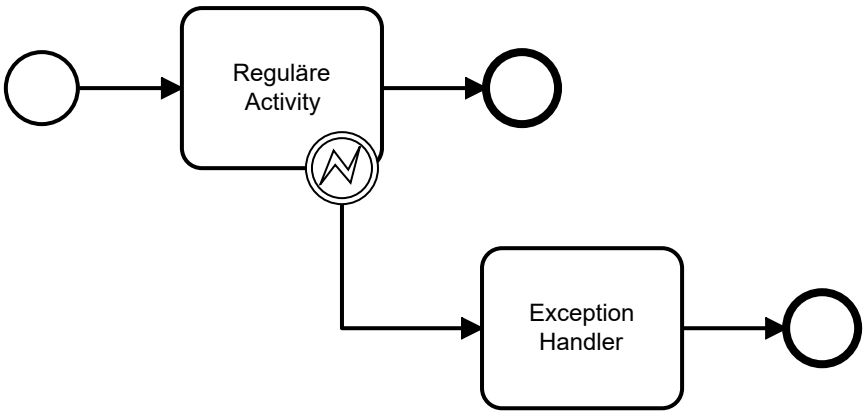


Abbildung 2.7: Exception Handling in BPMN. Exception Paths werden mittels Boundary Events definiert, die an Activities angehängt werden, nach [RW12]

- *External Event*: Workflows bilden einen Prozess der echten Welt ab. Da es aufgrund der Komplexität der echten Welt kompliziert und teilweise unmöglich ist den Prozess der echten Welt als Prozessmodell in einem Workflow akkurat umzusetzen, kann es zu Ungleichheiten zwischen den beiden Prozessen kommen. Entsprechend müssen Änderungen im Prozess der echten Welt an das System, bzw. den Software-Prozess weitergegeben und vorhandene Fehler im Software-Prozess angepasst werden.
- *Constraint Violations*: Elemente eines Prozesses, z. B. Activities, unterliegen gewissen Einschränkungen. So kann beispielsweise eine Einschränkung sein, dass der Versandprozess eines Bestellvorgangs die Adressdaten des Kunden benötigt, um den Versand in die Wege zu leiten. Falls diese fehlen, wird die Einschränkung verletzt und es kommt zur Exception.

Die Unterscheidung der Quellen von Exceptions ist wichtig, da daraus unterschiedliche Methoden zur Behandlung von Exceptions erwachsen. In BPMN werden Exceptions mittels Events definiert (siehe Abbildung 2.7). Exceptions können mithilfe von Events somit sowohl einzelnen Tasks als auch Subprozessen zugeordnet werden. Reichert et al. definieren drei Kategorien von Exception Handling Patterns, die hier aufgeführt sind:

- *Trying Alternatives Patterns*: In diesem Pattern werden alternative Pfade ausgeführt, wenn ein gewollter Prozess nicht ausgeführt werden kann. Dabei kann zwischen *Ordered Alternatives* und *Unordered Alternatives* Patterns unterschieden werden. Ordered Alternatives beschreiben eine vorgegebene Reihenfolge von alternativen Tasks, welche eingehalten werden muss. Unordered Alternatives können in arbiträrer Reihenfolge ausgeführt werden (siehe Abbildung 2.8).

- *Adding Behavior Patterns*: Hierbei sorgen die Exception Handler, dass zusätzliche Tasks ausgeführt werden. Wenn die zusätzlichen Tasks sofort ausgeführt werden, handelt es sich um *Immediate Fixing*. In dem Fall wird mit der Fortführung des Prozesses gewartet, bis die zusätzlichen Schritte abgearbeitet worden sind. Die Behandlung kann jedoch auch verzögert geschehen (*Deferred Fixing*). Dabei wird das aufgetretene Problem dokumentiert und kann

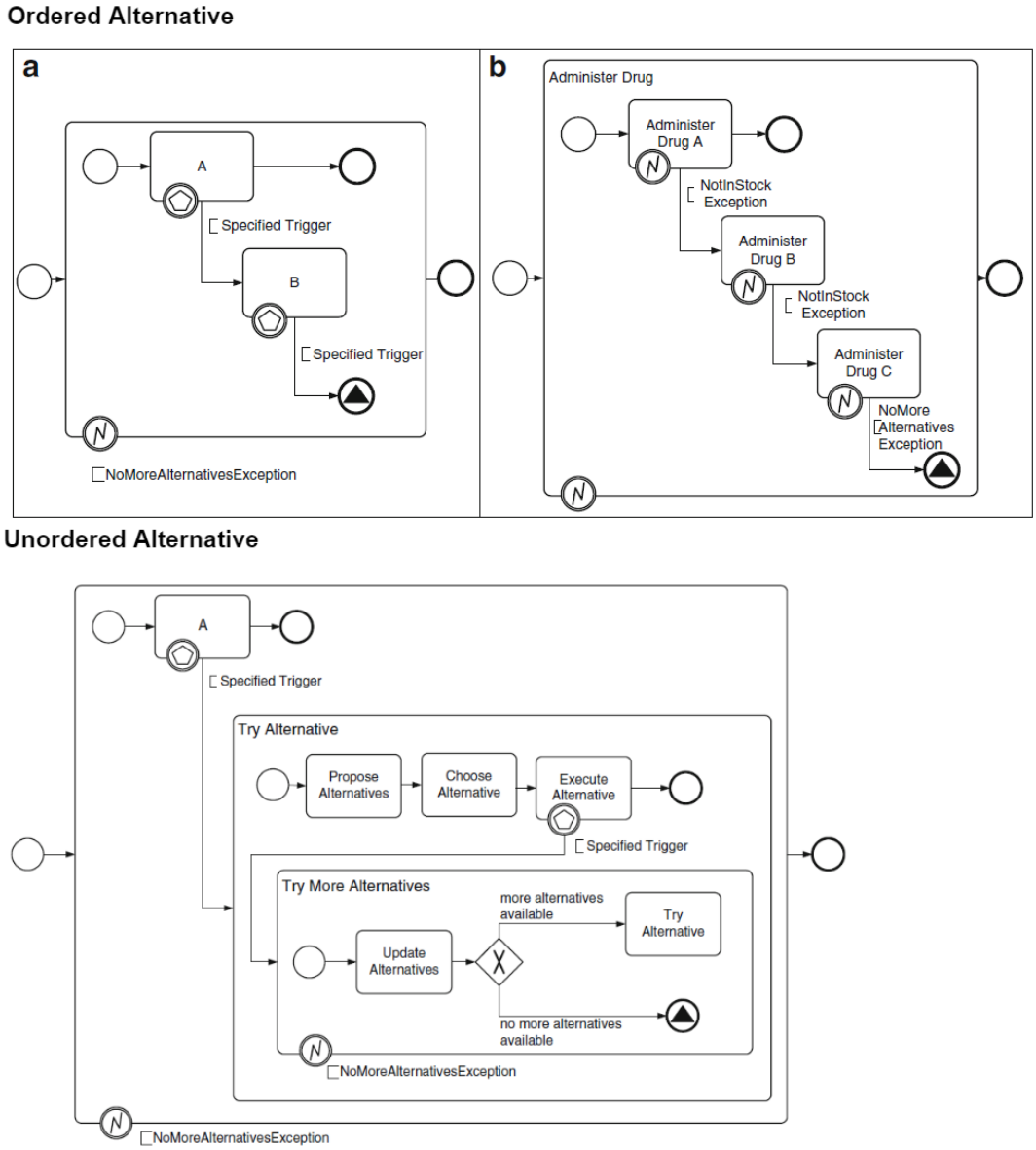


Abbildung 2.8: Beispiele von Trying Alternative Patterns. Ordered Alternatives haben eine festgelegte Reihenfolge, während Unordered Alternatives in beliebiger Reihenfolge ausgeführt werden können, aus [RW12]

unter Umständen zu einem späteren Zeitpunkt behandelt werden. Eine weitere Möglichkeit ist der *Retry*, wo die gewollte Activity direkt nochmal ausgeführt wird. Alternativ kann man die Ausführung der Activity auch mittels *Rework* verzögert wiederholen (siehe Abbildung 2.9).

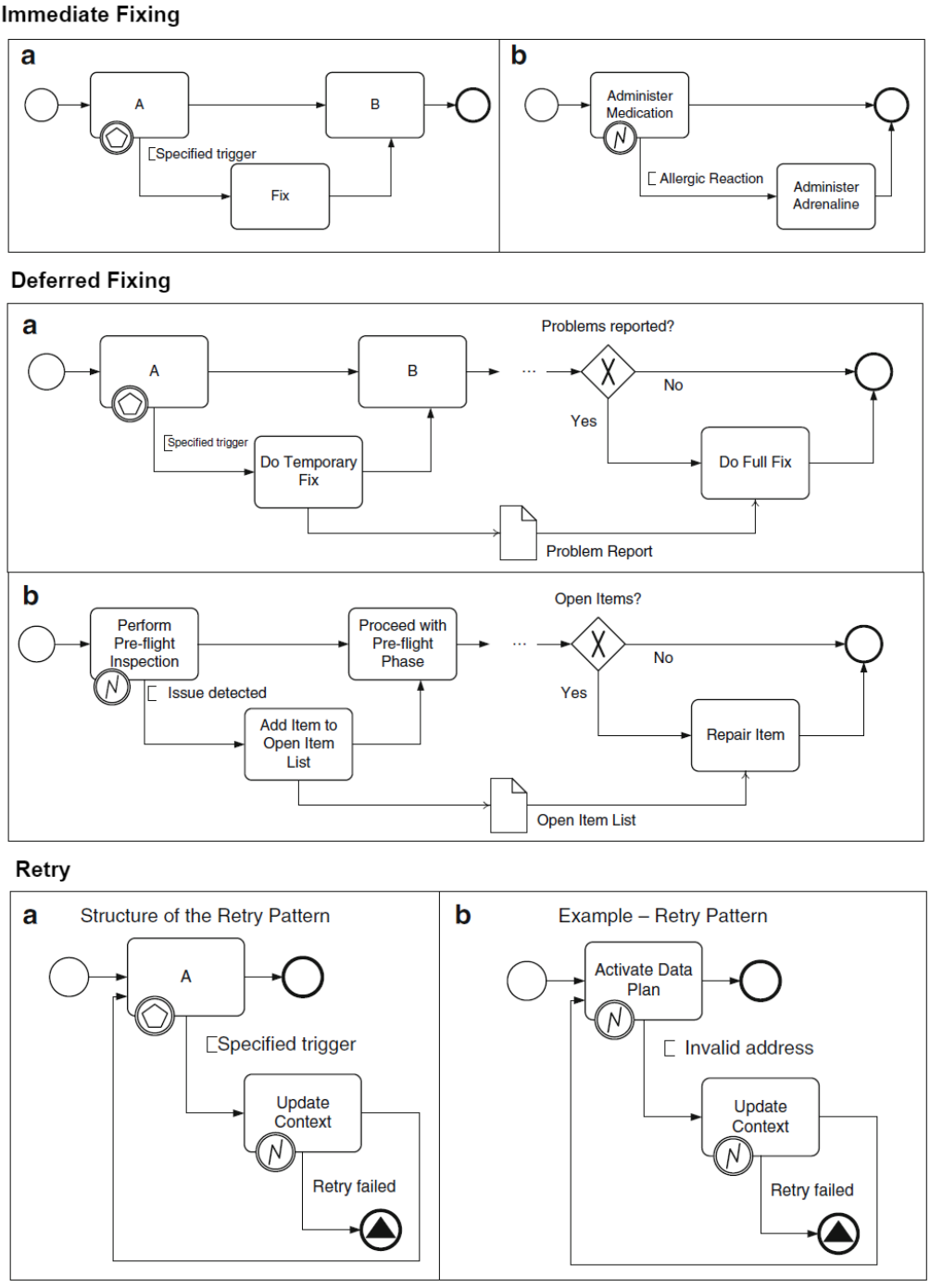


Abbildung 2.9: Beispiele von Adding Behavior Patterns. Immediate Fixing behandelt Exceptions sofort, Deferred Fixing verzögert die Behandlung der Exception und Retry versucht die Activity erneut auszuführen, aus [RW12]

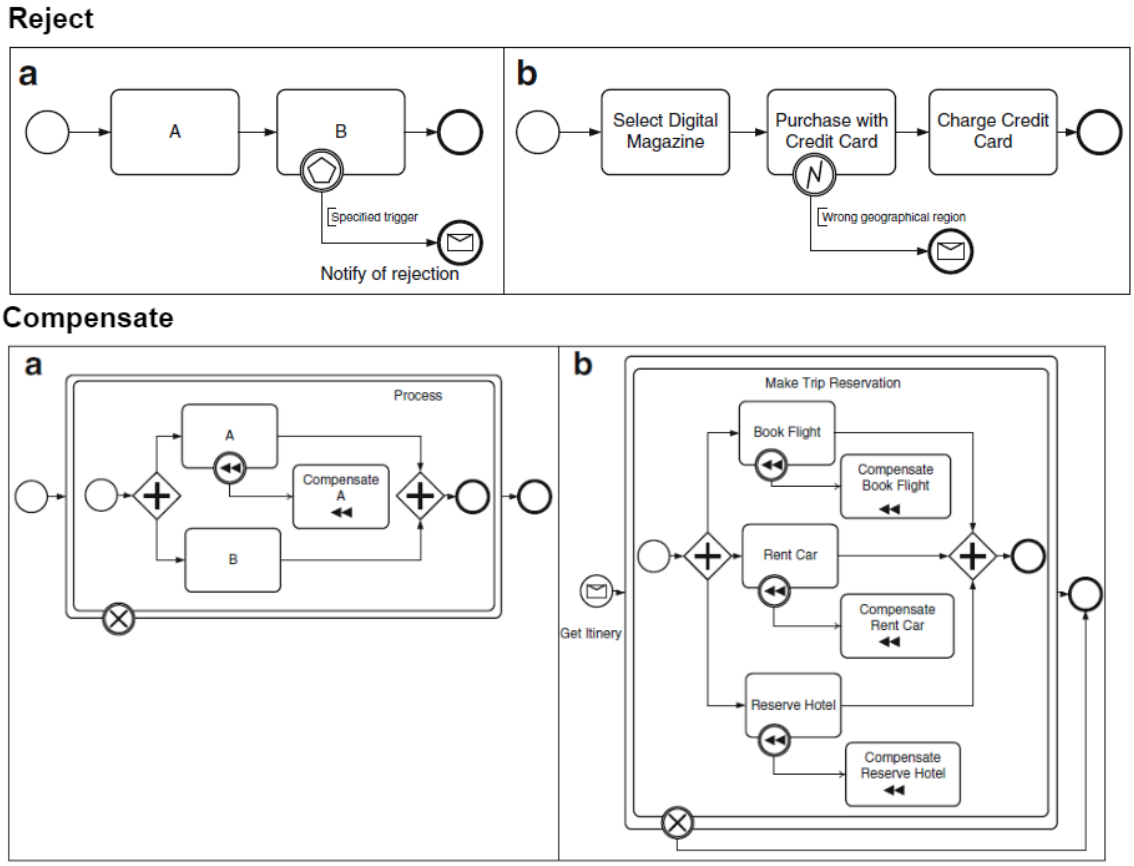


Abbildung 2.10: Beispiele von Canceling Behavior Patterns. Reject bricht die Ausführung komplett ab, während Compensate die Ausführung abbricht und bereits ausgeführte Arbeit rückgängig macht, aus [RW12]

- *Canceling Behavior Patterns*: Dabei wird die Ausführung des Prozesses beim Auftreten einer Exception unterbrochen. Falls lediglich die Ausführung komplett abgebrochen wird und keine weiteren Activities ausgeführt werden, spricht man vom *Reject*, wird bereits ausgeführte Arbeit rückgängig gemacht von *Compensate* (siehe Abbildung 2.10).

Eine komplette Übersicht über Quellen von Exceptions und dem Exception Handling lässt sich Abbildung 2.11 entnehmen.

2.7 Compensation Spheres

Compensation Spheres sind ein Konzept, welches Kompensationsverhalten für Gruppen von Activities definiert [LR00]. Wenn eine Activity fehlschlägt, müssen die vorhergehenden Activities rückgängig gemacht werden. Gruppen von Activities sollen also atomar ausgeführt werden. Im Normalfall werden Transaktionen für so etwas verwendet, jedoch soll es auch möglich sein nicht-transaktionale Activities zurückzusetzen. Wenn man beispielsweise eine Reise bucht, muss der Flug gebucht, das Auto gemietet und ein Zimmer reserviert werden. Alle drei Activities sind eng gekoppelt und müssen rückgängig gemacht werden, wenn einer der drei Activities nicht klappt.

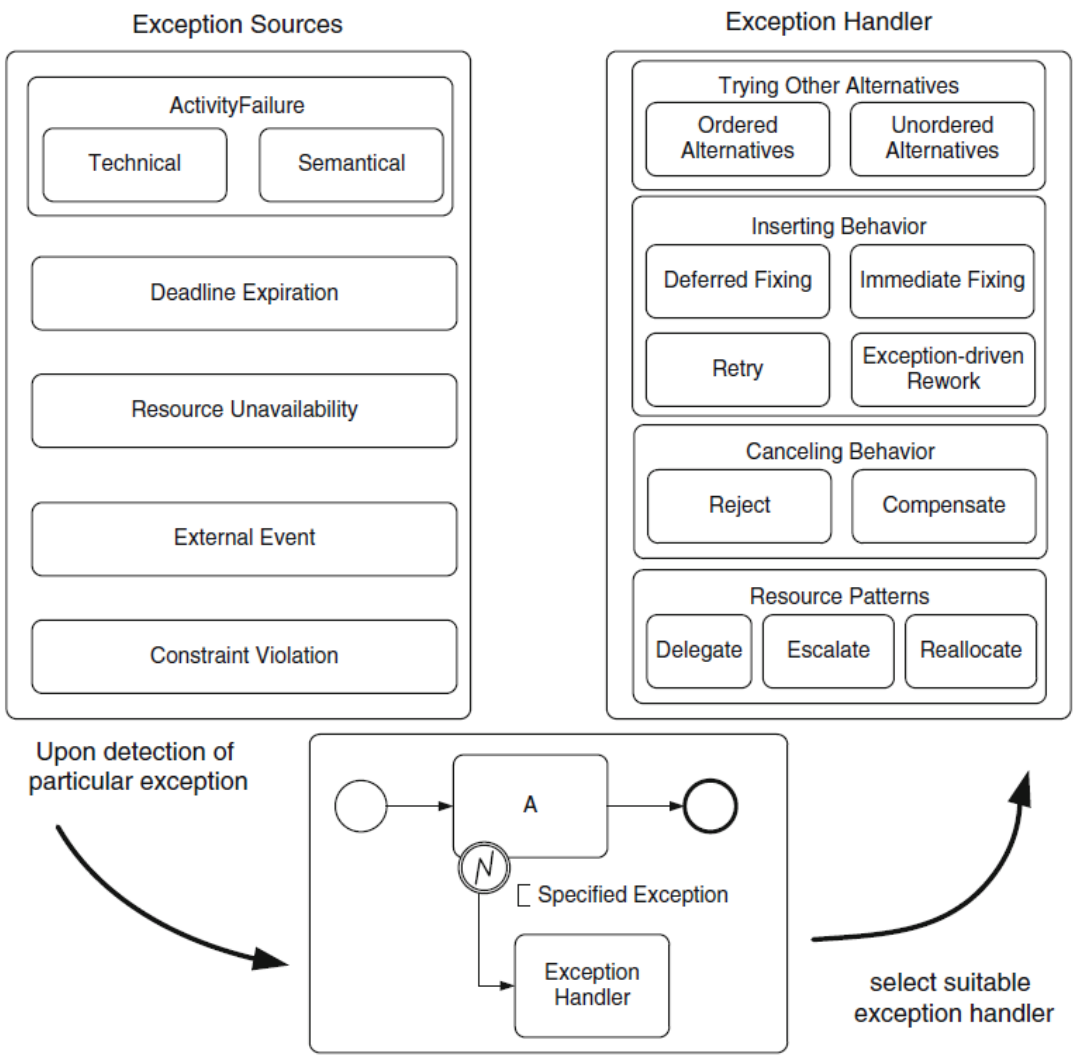


Abbildung 2.11: Übersicht über Exception Sources und Handlers, aus [RW12]

Um dies sicherzustellen wird eine Gruppe von Arbeit in einer Sphere zusammengefasst, welcher die Ausführung aller Activities garantiert. Diese Sphere bekommt einen *Compensation Handler* zugeordnet. Auch einzelne Activities können eine Compensation zugeordnet bekommen. Wenn nun eine Activity nicht klappt, werden alle Compensation Handler rückwärts, entgegen der Ausführungsreihenfolge der Activities, ausgeführt. In Abbildung 2.12 sind Beispiele von Compensation Spheres und einzelnen Compensations in einem Schaubild dargestellt.

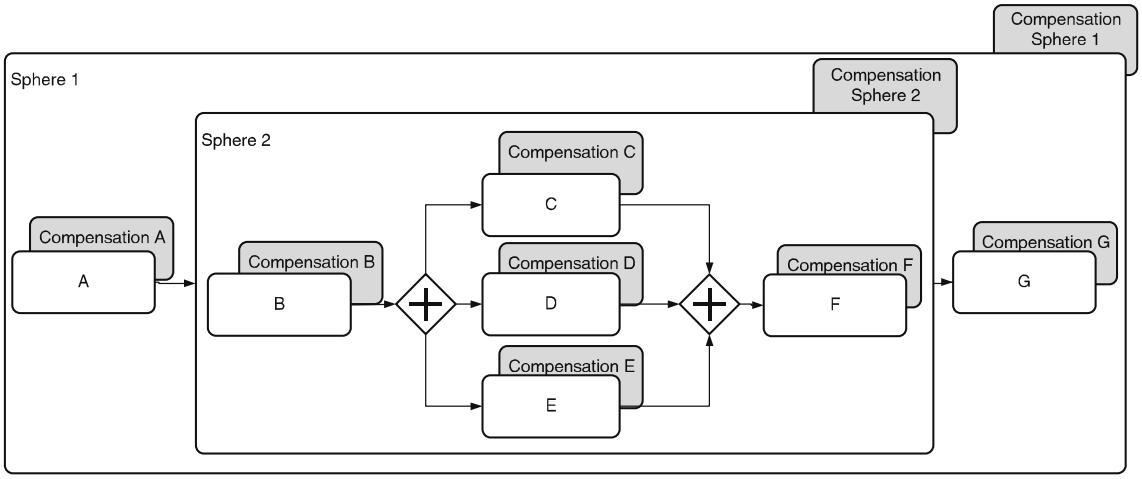


Abbildung 2.12: Beispiel von Compensations und Compensation Spheres. Compensations werden an einzelne Activities angehängt. Compensation Spheres können eine Gruppe von Activities umfassen und im Fehlerfall kompensieren, aus [RW12]

3 Verwandte Arbeiten

Dieses Kapitel stellt Forschungsarbeiten vor, die thematisch ähnlich zu der Arbeit dieser Thesis sind und die Entwicklung des Konzepts beeinflusst haben.

3.1 Kontextsensitive Systeme

Adaptive Workflow-Systeme interagieren mit ihrer Umwelt. Ein weiterer Begriff, der häufig synonym für Umwelt benutzt wird, ist *Kontext*. Es gibt eine Vielzahl von Definitionen für den Begriff Kontext. Eine der meist zitierten Definitionen stammt dabei von Dey [Dey01]: "Context is any information that can be used to characterize the situation of an entity". Entity wird folgendermaßen von ihm definiert: "An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Dies ist eine sehr umfangreiche Beschreibung und umfasst, insbesondere in Bezug auf den User, also den Nutzer, auch Informationen über Elemente, die unzureichend objektiv beschrieben und gemessen werden können, wie beispielsweise menschliches Verhalten [Dou04]. Da die Auflösung dieses Konflikts nicht im Umfang dieser Thesis liegt, werden für die Definition von Kontext nur objektiv messbare Informationen angenommen. Entsprechend sind kontextsensitive Systeme solche, die ihre Umwelt (ihren Kontext) überwachen und messen und, abhängig von diesem Kontext, von menschlicher Intervention unabhängige, Handlungen ausführen können [IAC16]. Computersysteme überwachen ihre Umwelt mittels Sensoren. So kann beispielsweise ein smarter Thermostat die Temperatur eines Zimmers messen und je nach Ergebnis die Temperatur ändern. Jedoch muss der Thermostat wissen, nach welchen Regeln er auf die Temperatur reagieren soll. Auch diese müssen in einem Prozessmodell definiert werden. Selbst wenn man nur die objektiv messbaren Informationen als Grundlage nimmt, wird bei Systemen die nicht vollständig triviale Aufgaben ausführen, die Anzahl der möglichen Kontexte unüberblickbar. Laut Greenberg [Gre01] ist für den Systementwickler daher die Aufgabe, dass er möglichen Kontexte, die Beschreibungen der Informationen die einen Kontext akkurat bestimmt, sowie die Aktionen, die in einem bestimmten Kontext ausgeführt werden sollen, definieren soll. Ein kontextsensitives System sollte die Modellierung solcher Informationen unterstützen.

3.1.1 SitME

Breitenbücher et al. [BHK+15] haben eine Erweiterung für Workflows namens Situation-Aware Workflow Modelling Extension (SitME) entworfen, welche die Einbindung von Kontext im Design von Workflows ermöglicht. Sie definieren Kontext als Situationen, welche auftreten können. Dafür haben sie einen Event-Typ namens *Situation Event* konzipiert. Dieses besteht aus einem Situationsnamen und einem einzigartigen Identifier, welcher das Objekt für welches das Event gilt, definiert. Die Situationen werden automatisch vom dazugehörigen System überwacht. Wenn nun eine Situation auftritt, dann wird das Situation Event getriggert und die ausgehenden Kontrollflüsse

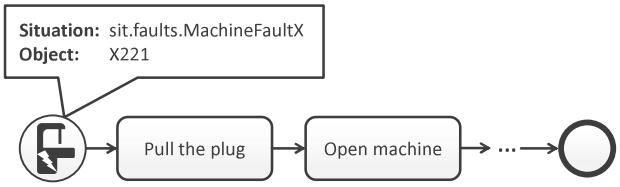


Abbildung 3.1: Ein Situation Event des SitME-Ansatzes. Das Situation Event wird vom System getriggert, wenn die dazugehörige definierte Situation eintritt, aus [BHK+15]

werden aktiviert (Siehe Abbildung 3.1). Des Weiteren haben sie die *Situational Scopes* entwickelt. In Abbildung 3.2 ist ein solcher dargestellt. Diese gruppieren mehrere Activities zusammen, welche genau dann ausgeführt werden, wenn eine dafür spezifizierte Situation eintritt. Diese Situational Scopes werden in den Kontrollfluss eingebunden. Eine Situation, welche an die Situational Scopes angebunden wird, besitzt die gleichen Attribute wie Situation Events. Zusätzlich wird jedoch definiert, was passieren soll, wenn der Kontrollfluss den Situational Scope erreicht und die Situation noch nicht eingetreten ist. Als Attribut dafür kann entweder eine Wartezeit in Form des Attributs "*Wait([Duration])*" oder der direkte Abbruchbefehl "*Abort*" gesetzt werden. Wenn der Wartebefehl gesetzt wird, dann wartet der Kontrollfluss die angegebene Zeit "*Duration*" darauf, dass die Situation eintritt. Wenn sie eintritt, wird der Situational Scope ausgeführt und der normale Kontrollfluss wird weiter ausgeführt. Falls die Situation nicht in der angegebenen Zeit auftritt, wird der Prozess mit einem Fehlerfall abgebrochen. Dieses Konzept wurde für die Workflow-Ausführungssprache Business Process Execution Language (BPEL) umgesetzt und die Erweiterung der Sprache namens "SITME4BPEL" entwickelt. Mithilfe dieser lassen sich situationsabhängige Workflows definieren und dann mittels eines Transformationsalgorithmus in ein standardkonformes BPEL-Modell umwandeln.

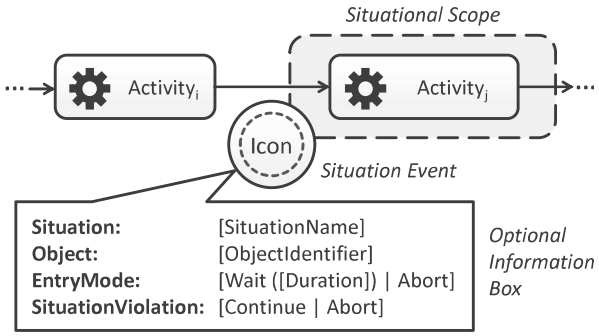


Abbildung 3.2: Ein Situational Scope des SitME-Ansatzes mit den dazugehörigen Attributen, aus [BHK+15]

3.1.2 SitOpt

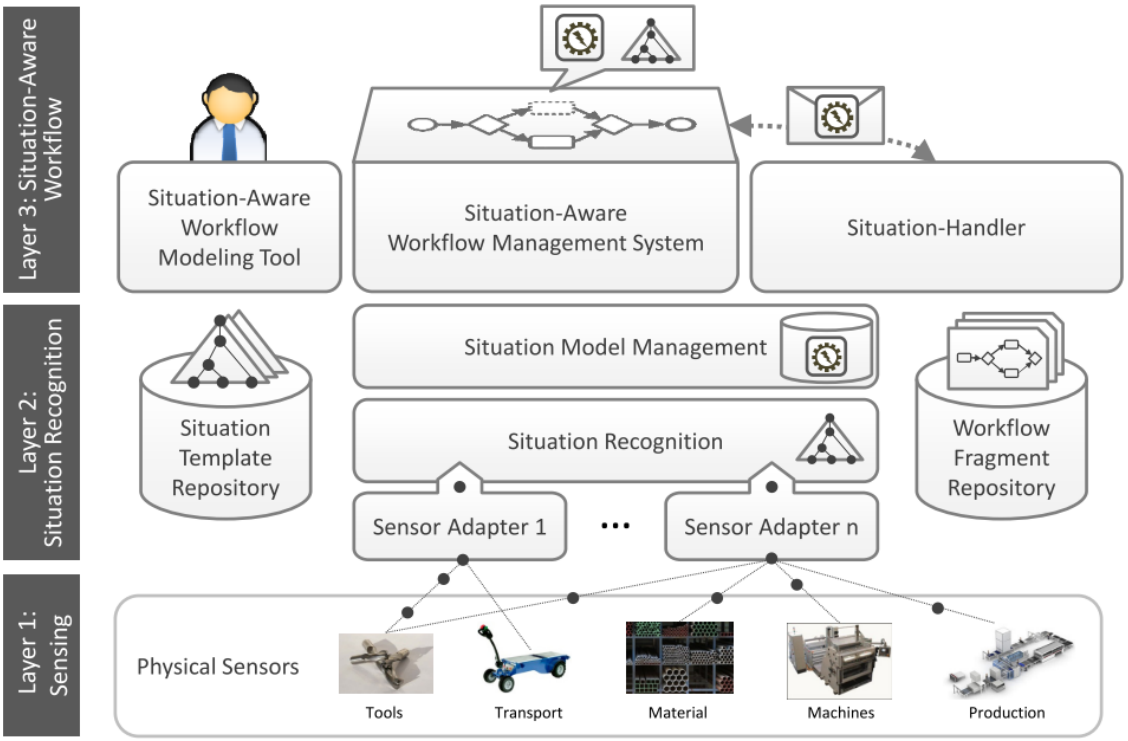


Abbildung 3.3: Darstellung der Systemarchitektur von SitOpt, welche aus den drei Ebenen “Sensing“, “Situation Recognition“ und “Situation-Aware Workflow“ besteht, aus [WSBL15]

SitOpt ist ein Ansatz, welcher es Anwendungen erlaubt sich nicht nur autonom, sondern auch situationsabhängig an die Umgebung anzupassen. Es wurde an der Universität Stuttgart entwickelt. Die Systemarchitektur zeigt Abbildung 3.3. SitOpt besteht aus drei Ebenen: der Sensorebene, der Situationserkennungsebene und der situationsabhängigen Workflow-Ebene. In der Sensorebene messen Sensoren die Umgebung und geben diese Daten weiter an die Situationserkennungsebene. In dieser werden anhand von gespeicherten Situationsmodellen Situationen erkannt. Diese Daten werden von der situationsabhängigen Workflow-Ebene wiederum verwendet, um die zu der erkannten Situation passenden Workflows anzupassen und auszuführen [WSBL15].

3.2 Transformation von Choreographie zu Kollaboration

Die Transformation eines Choreographiemodells in ein Kollaborationsmodell ist nicht trivial. Choreographiemodelle haben eine globale Sicht auf eine Interaktion, d. h. sie definieren die Sequenz der globalen Interaktion. Kollaborationsmodelle haben hingegen eine lokale Sicht auf ihren Prozess mit aus- und eingehenden Interaktionen als Teil ihres Prozesses. Bei einer Transformation muss also sichergestellt werden, dass die lokale Durchführbarkeit des Prozesses gegeben bleibt. Die Struktur des Choreographiemodells bleibt in den einzelnen Teilnehmern im Kollaborationsmodell

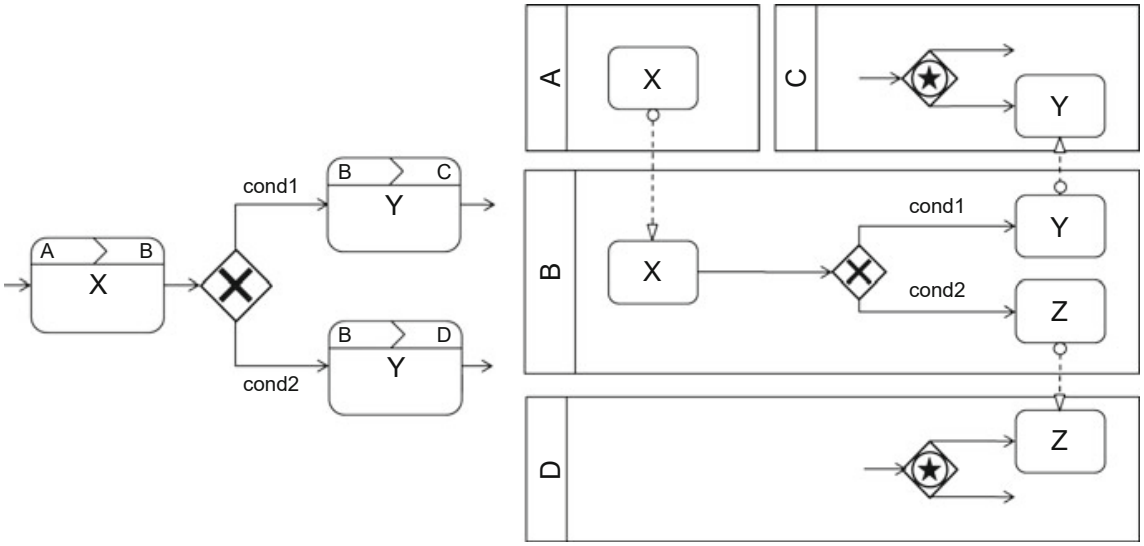


Abbildung 3.4: Darstellung einer Choreographie-Transformation. Links sind die Choreographieinteraktionselemente (in BPMN wären das Choreography Tasks), rechts sind die tranformierten Kollaborationspartner. Die Struktur des Choreographieprozesses bleibt in Kollaborationsdiagrammen erhalten, aus [BHF10]

grundsätzlich erhalten. Ein besonderes Augenmerk muss dabei auf die Transformation von Gateways gelegt werden [BHF10]. Abbildung 3.4 zeigt beispielhaft, wie ein exklusives OR-Gateway in einer Choreographie in den einzelnen Partnerprozessen umgesetzt werden muss. Partner B kann hier nach dem Nachrichtenempfang von Partner A den Kontrollpfad auf zwei verschiedene Arten fortführen: Entweder sendet er eine Nachricht an Partner C oder eine Nachricht an Partner D. Diese exklusive Entscheidung muss hierbei im Prozess von Partner B erhalten bleiben. Partner C und D hingegen benötigen ein anderes Konstrukt: Sie müssen die Nachricht von Partner B empfangen und genau dann, wenn die Nachricht empfangen wird, diesen Kontrollpfad beschreiten. Da dies nur eine lokale Betrachtung des Empfangs der Nachricht darstellt, müssen die Partner C und D die Möglichkeit erhalten, einen alternativen Pfad zu nehmen, wenn der Prozess einen alternativen Kontrollfluss anstelle des Nachrichtenempfangs vorsieht. Dies kann man durch ein Event-based Gateway sicherstellen. Falls kein alternativer Kontrollfluss vorgesehen ist, kann dieses Gateway auch weggelassen werden [BHF10].

Bischoff et al. [BFR19] haben beschrieben, wie man Choreographiemodelle generiert und in Kollaborationsmodelle transformiert. Auch sie haben Verzweigungen in Modellen als das größte Problem für die Transformation identifiziert. Zunächst transformieren sie in ihrem Algorithmus für jeden Teilnehmer einer Choreographie die Choreography Tasks in Send, bzw. Receive Tasks, abhängig davon, ob die der Teilnehmer der initiierende Interaktionspartner ist oder nicht. Da nicht jeder Interaktionsteilnehmer in jedem Alternativpfad an einer Interaktion beteiligt ist, wird bei Gateways überprüft, ob der Teilnehmer in den, zum Gateway gehörenden, Pfaden beteiligt ist. Das Gateway wird im Prozess eines bestimmten Interaktionspartners entsprechend nur dann gesetzt, wenn ein Choreography Task dieses Partners in den Gateway-Pfaden gefunden wird.

3.3 Service Interaction Patterns

Weske et al. [Wes19] beschreibt mehrere Patterns, wie man Interaktionen darstellen kann. Die für diese Thesis relevanten werden hier kurz vorgestellt:

- *Send/Receive*: Das einfachste Pattern ist das Send/Receive Pattern. Ein Teilnehmer sendet dabei eine Anfrage an einen anderen Teilnehmer. Dieser antwortet auf die Anfrage, indem er eine Nachricht zurücksendet. Da in einem Prozess theoretisch mehrere Nachrichten gleichzeitig gesendet und empfangen werden können, müssen diese Nachrichten aneinander gebunden werden (siehe Abbildung 3.5).

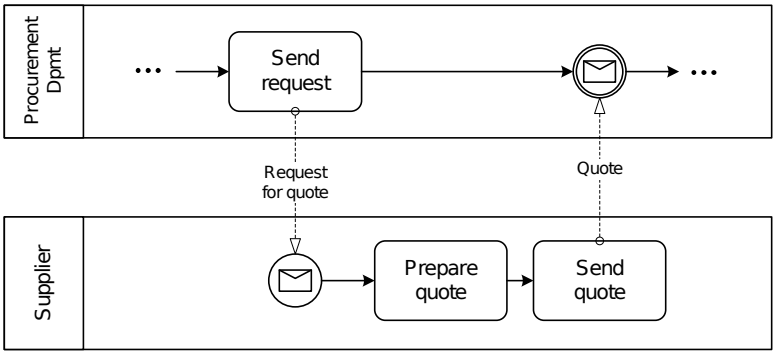


Abbildung 3.5: Darstellung des Send/Receive Patterns. Hierbei sendet ein Teilnehmer einem anderen Teilnehmer eine Nachricht, anhand welcher ein Prozess dieses Teilnehmers angestoßen wird, aus [Wes19]

- *Racing Incoming Messages*: Dieses Pattern stellt einen Wettlauf von Nachrichten dar. Dabei wartet ein Teilnehmer auf den Empfang einer Nachricht. Mehrere Teilnehmer können diesem eine Nachricht schicken, jedoch wird der Teilnehmer nur die erste Nachricht, die er erhält, annehmen. Dieses Pattern kann beispielsweise verwendet werden, wenn dem empfangenden Teilnehmer egal ist, von wem er ein Angebot für eine Ware erhält und ihn nur interessiert, dass ein Angebot eingeht (siehe Abbildung 3.6).

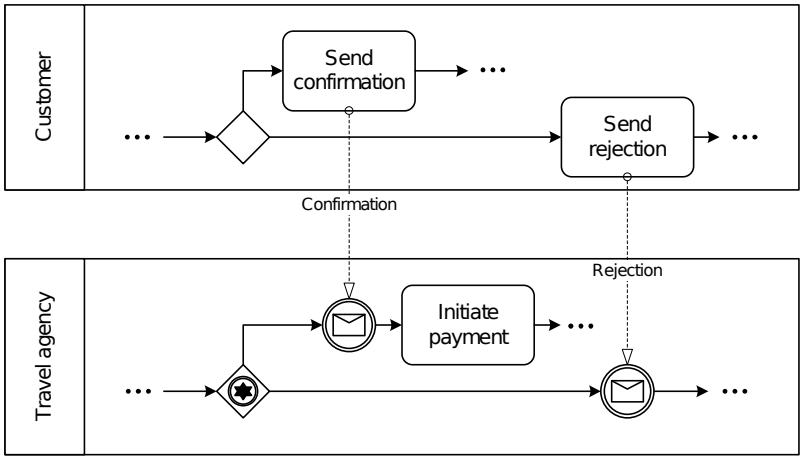


Abbildung 3.6: Darstellung des Racing Incoming Messages Patterns. Hier konkurrieren mehrere Nachrichten miteinander darum als Erstes bei einem Teilnehmer anzukommen. Der Teilnehmer wartet auf Nachrichten bis die erste Nachricht ankommt und hört danach auf zu warten, aus [Wes19]

- *One-To-Many Send*: Bei diesem Pattern sendet ein Teilnehmer mehrere Nachrichten an andere Teilnehmer gleichzeitig. Dies könnte etwa relevant sein, wenn der Teilnehmer ein Rundschreiben an eine Zahl von Teilnehmern senden will. Die Zahl der Teilnehmer kann dabei auch erst zur Laufzeit feststehen (siehe Abbildung 3.7).

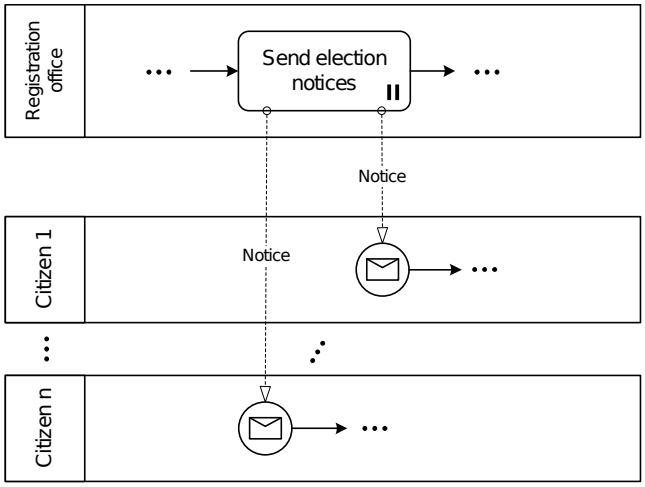


Abbildung 3.7: Darstellung des One-To-Many Send Patterns. Hier sendet ein Teilnehmer Nachrichten an mehrere andere Teilnehmer gleichzeitig, aus [Wes19]

- *One-From-Many Receive*: Hier empfängt ein Teilnehmer Nachrichten von mehreren Teilnehmern. Im Gegensatz zum Racing Incoming Messages Pattern hört er jedoch nicht auf mit dem Warten, sobald eine Nachricht angekommen ist. Daher muss ein Trigger für den Abbruch definiert werden, häufig ist dies ein Timer (siehe Abbildung 3.8).

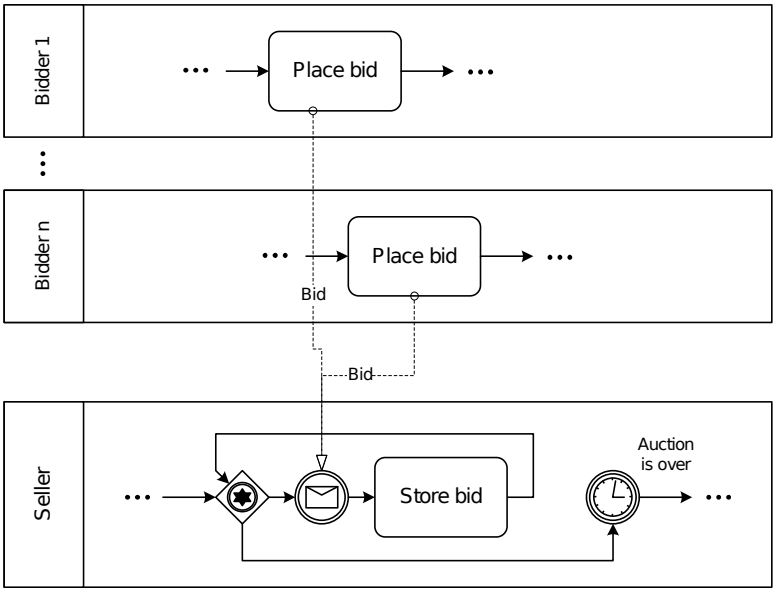


Abbildung 3.8: Darstellung des One-From-Many Receive Patterns. Hierbei wartet ein Teilnehmer auf die Nachrichten mehrerer anderer Teilnehmer, aus [Wes19]

4 Use Case

Das fortlaufende Szenario in dieser Thesis ist ein fiktionaler Prozess in einem Smart Home-Kontext, welcher dem Nutzer einen Kaffee kochen soll. Dieser ist vollautomatisiert und es wird davon ausgegangen, dass verschiedene Sensoren zur Verfügung stehen, welche etwa feststellen können, ob beispielsweise Kaffeebohnen zur Verfügung stehen oder nicht. Ebenso geht das Beispiel davon aus, dass es Möglichkeiten zur mechanischen Interaktion mit Gegenständen gibt, wie etwa ein Roboterarm. Der Gesamtprozess als Choreographiediagramm ist in Abbildung 4.1 abgebildet.

Wenn der Nutzer den Prozess zum Kaffeekochen anstößt, überprüft dieser, ob in der Kaffeemaschine Kaffeebohnen vorhanden sind und ob die Kaffeemaschine funktionsfähig ist. Wenn ja, wird der Kaffee gekocht. Wenn keine Kaffeebohnen vorhanden sind, wird der “Kaffee-Kaufen“-Prozess angestoßen. Dieser sucht bei einem Internethändler den passenden Kaffee heraus, bestellt diesen und registriert den Paketboten bei der intelligenten Klingel, welche für ihn bei Lieferung eine einmalige Zutrittsautorisierung ausstellt. Außerdem benachrichtigt er das Verstauungssystem, welcher mittels mechanischem Arm den Kaffee nach Lieferung automatisch in die Maschine füllt. Zum Zweck des Verstauens schickt das Verstauungssystem eine Nachricht an einen Roboterarm, welcher diese Aufgabe ausführt. Sollte die Lieferung nicht innerhalb von 5 Tagen erfolgen, sorgt ein Timeout dafür, dass der Nutzer über den Fehlschlag informiert wird.

Ein anderer Pfad wird ausgeführt, wenn die Kaffeemaschine funktionsuntüchtig ist. In diesem Fall wird der Reparaturprozess angestoßen, welcher automatisch einen Termin bei einem Reparateur ausmacht. Danach wird der “Tee-Kochen“-Prozess initiiert. Dieser wartet darauf, dass der Teekocher bereit ist, Tee zu kochen. Sobald dies der Fall ist, schickt der Prozess dem Teekocher den Ausführbefehl. Falls der Teekocher nicht innerhalb von 5 Minuten bereit ist, geschieht ein Timeout und der Kontrollfluss wird zurückgegeben an den vorherigen Prozess und eine Nachricht über den Fehlschlag des Prozesses wird an den Nutzer geschickt.

Dieses Szenario verdeutlicht nochmals die Problematik: In menschlicher Sprache beschrieben erscheint es nicht sonderlich komplex. Das BPMN-Kollaborationsdiagramm in Abbildung 4.1, welches sogar nur die Interaktion zwischen den einzelnen Teilnehmern darstellt, ist hingegen schon äußerst unübersichtlich und umständlich zu modellieren. Entsprechend wird im folgenden Kapitel ein Konzept vorgestellt, welches in der Lage ist, Situationen und die dazugehörigen Ausführungspfade zu definieren und dieses Modell in ein Choreographiemodell zu transformieren.

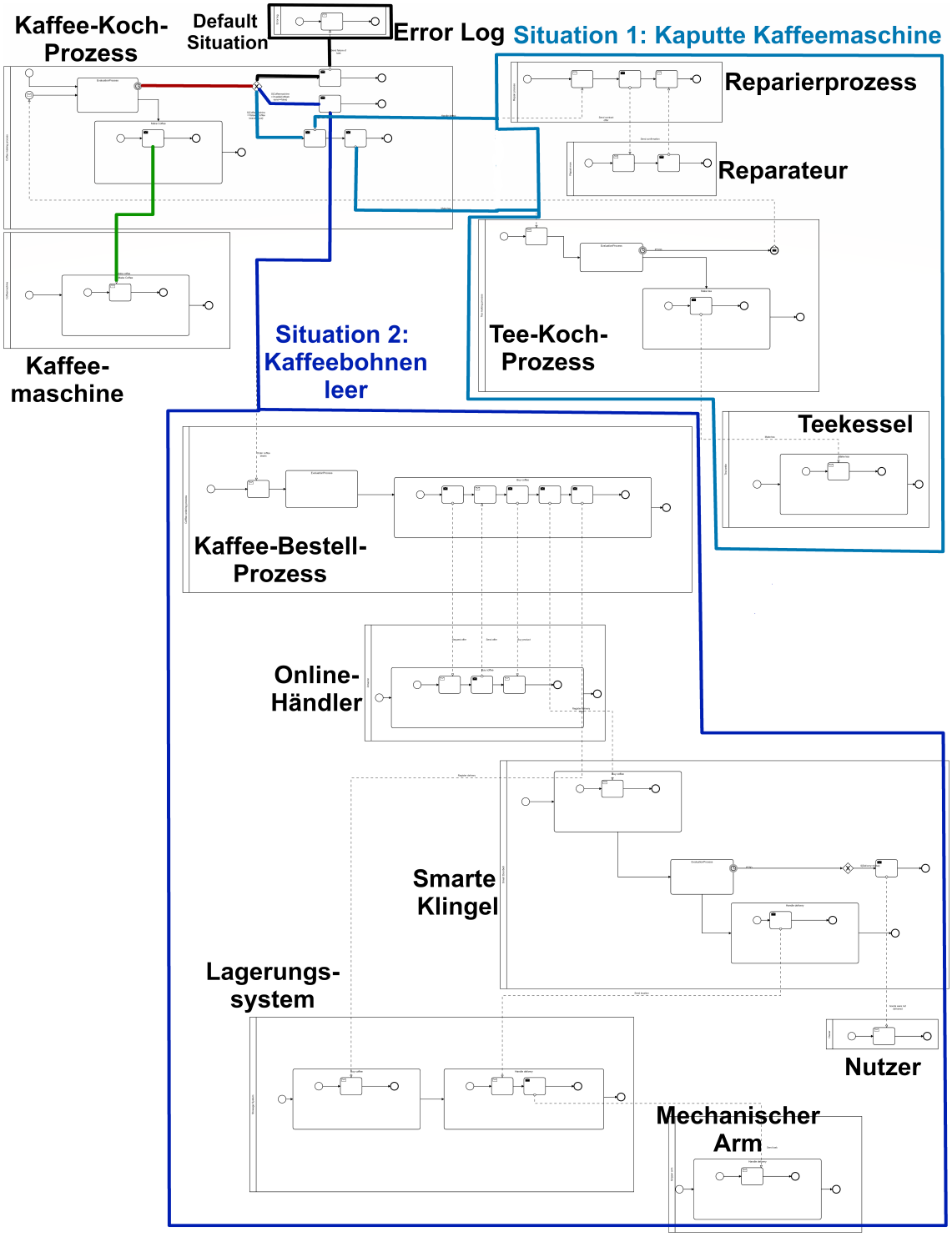


Abbildung 4.1: Kollaborationsdiagramm eines “Kaffee-Kochen“-Prozesses mitsamt allen möglichen Fehlerbehandlungen für bestimmte Situationen. Der Verständlichkeit halber sind die verschiedenen Situationen und die dazugehörigen Ausführungspfade mittels farblichen Markierungen hervorgehoben

5 Konzept

Dieses Kapitel stellt den grundsätzlichen Entwicklungsansatz, das entwickelte Modellierungskonzept, sowie die Ideen und Entscheidungen, die in beides eingeflossen sind, vor. Da in vorherigen Kapiteln die Problematiken der Modellierung komplexer, situationsabhängiger Systeme mittels Choreographien in Workflow-Sprachen wie BPMN bereits ausführlich behandelt wurde, werden sie hier lediglich nochmals kurz zusammengefasst.

Ein großes Problem bei der Modellierung solcher Systeme ist die Diskrepanz zwischen dem, was modelliert werden soll, also Situationen, und dem, was die Workflow-Sprachen an Konzepten zur Verfügung stellen. So lassen sich in BPMN mittels Aktivitäten und Entscheidungen zwar Situationen modellieren, jedoch nur implizit, wodurch das Hinzufügen, bzw. Ändern von Situationen nur mit erheblichem Aufwand und einer hohen Fehleranfälligkeit möglich ist. Die grundsätzliche Idee des Ansatzes dieser Thesis ist Situationen zu einem expliziten Element der Modellierung zu machen und dieses Modell in einen ausführbaren Zustand zu bringen.

Entsprechend basiert der Ansatz dieser Thesis auf der Modellierung von BPMN-Choreographiediagrammen mittels Situationen. Die Choreographien werden für einzelne Situationen modelliert und später in ein verknüpftes ausführbares Modell umgewandelt. Da Situationen in BPMN-Modellen implizit dargestellt werden können, ist es nicht sinnvoll eine komplett eigenständige und von bestehenden Standards losgelöste Lösung für die Ausführung der Modelle zu entwickeln. Darum gehört auch die Transformation des situationsbezogenes Modells in ein standardkonformes BPMN-Kollaborationsdiagramm zum Ansatz. Durch die Transformation wird dabei ein Collaboration Model, also ein Modell, welches die, für die Interaktion relevanten, Tasks der einzelnen Teilnehmer darstellt, erstellt. Diese lokalen Teilprozesse der Teilnehmer werden dann mit technischen Details ausgefüllt. Dieses Modell stellt also eine Art Prozessskelett dar, welches durch die Anreicherung von Aktivitäten durch einen Modellierer zu einem ausführbaren Prozessmodell weiterentwickelt wird. Der Modellierungsansatz besteht entsprechend aus den folgenden vier Schritten, welche auch nochmals in Abbildung 5.1 dargestellt sind:

- Erstelle situationsbezogenes Modell: Mithilfe des situationsbasierten Modellierungskonzepts wird ein Modell erstellt.
- Transformation in ein Kollaborationsdiagramm: Das erstellte Modell wird in ein standardkonformes BPMN-Kollaborationsdiagramm transformiert. Dabei werden die Strukturen und Attribute des Situationsmodells mithilfe von Vorlagen in passende Strukturen des Kollaborationsdiagramms umgewandelt.
- Anreicherung mit interner Prozesslogik: Das erstellte Prozessskelett wird um technische Aktivitäten ergänzt, welche für die korrekte Ausführung der Prozesse notwendig sind.
- Ausführung mittels Workflow-Engine: Das fertige Modell wird auf einer oder mehreren Workflow-Engine(s) ausgerollt und ausgeführt.

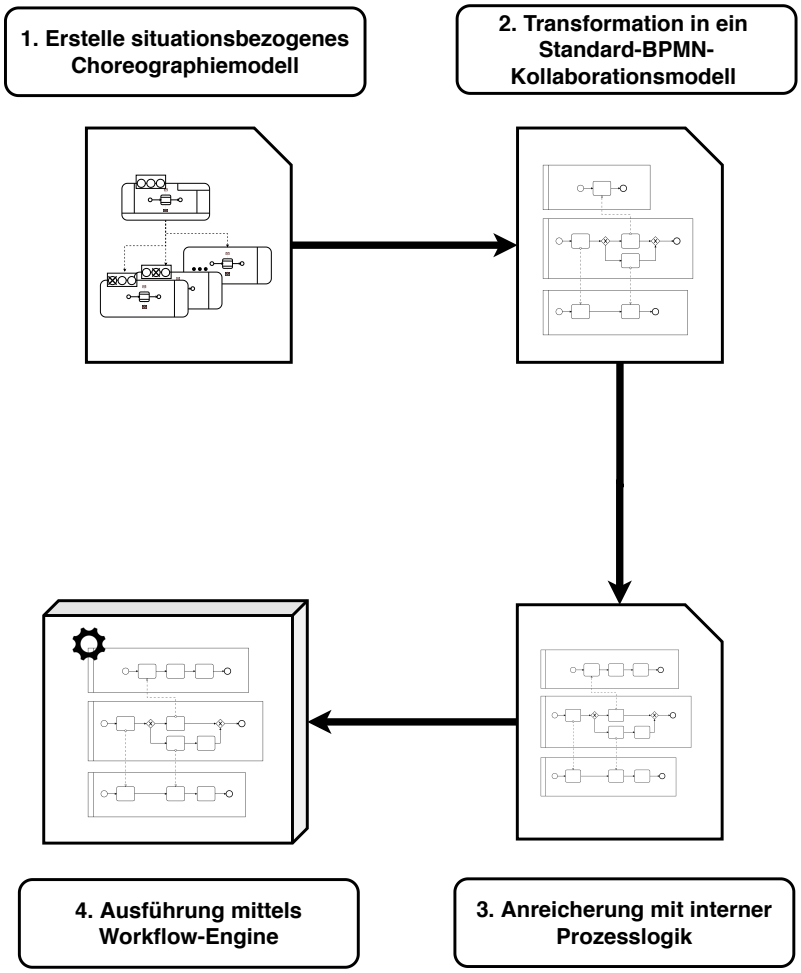


Abbildung 5.1: Gesamtübersicht über den ChorSitME-Ansatz

Dieser Ansatz orientiert sich am Ansatz, der von Breitenbücher et al. [BHK+15] entwickelt wurde. Dieser Ansatz wird SitME genannt, was für "Situation-Aware Workflow Modelling Extension" steht. Da der Ansatz dieser Thesis sich am SitME-Ansatz orientiert, wird ihm der Name "Choreography-based Situation-Aware Workflow Modelling Extension", kurz "ChorSitME", gegeben.

5.1 Modellierungskonzept mit Situationen

Zunächst wird das entwickelte Modellierungskonzept vorgestellt. Dazu werden die einzelnen Elemente des Konzepts, sowie ihre Zusammenhänge beschrieben. Außerdem wird beschrieben, wie die Elemente im transformierten Kollaborationsmodell umgesetzt werden.

5.1.1 Situational Scopes

Um Situationen zu modellieren, folgt das Modellierungskonzept dieser Thesis einem ähnlichen Ansatz wie Breitenbücher et al. [BHK+15]. Ein Konzept, welches adaptiert wurde, sind *Situational Scopes*. Außerdem unterstützt das Konzept die von Greenberg [Gre01] definierten Schritte zur Situationsmodellierung:

- Die einzelnen Situationen definieren.
- Die Elemente, die diese Situationen akkurat darstellen, definieren.
- Die Aktionen, welche beim Eintreten der Situation ausgeführt werden sollen, definieren.

Im Folgenden werden Situational Scopes definiert die in Abbildung 5.2 dargestellt werden. Ein Situational Scope bildet eine Situation der echten Welt ab. Dafür besitzt ein Situational Scope zwei Attribute, nämlich *Name* und *Situationselemente*. Diese Situationselemente sind einzelne Werte, die gelten müssen, um eine Gesamtsituation darzustellen. Situationselemente bestehen aus einem *Situationsnamen* und einem boolschen *Situationselementwert*. Die einzelnen Situationselemente werden bei Ausführung evaluiert und geprüft, ob der Scope und die darin enthaltenen Aktivitäten ausgeführt werden dürfen. Es ist auch möglich die Situation während der Ausführung zu evaluieren. In dem Fall wird, falls die Situationen während der Ausführung plötzlich nicht mehr gelten, die Ausführung unterbrochen und ein oder mehrere Alternativpfad(e) ausgeführt.

Um festzulegen, wie eine Situation evaluiert wird, werden *Entry Conditions* und *Running Compensate Conditions* verwendet. Eine Entry Condition definiert was passiert, falls die Situation nicht gilt, sobald der Kontrollfluss zu dem Punkt der Situation kommt. Die Running Compensate Condition definiert was passiert, wenn die Situation während der Ausführung des Prozesses nicht mehr gilt.

Alternative Pfade werden durch sogenannte *Adapt*-Pfade definiert. Demgegenüber stehen *Continue*-Pfade, die definieren wie der Standardkontrollfluss ausgeführt wird. Ein Continue-Pfad führt dabei zur nächsten Standardsituation. Um einen Situational Scope auszuführen, wird darin ein Choreographieprozess definiert, welcher den Ablauf darstellt, welcher in der geltenden Situation die relevanten Schritte ausführt. Falls die Situation nicht gilt, wird unter den angehängten Adapt-Pfaden nach einer oder mehreren passenden Alternativsituation(en) gesucht und diese ausgeführt. Diese Alternativsituationen werden wiederum durch Situational Scopes dargestellt, welche Situationselemente mit gleichem Namen aber mit unterschiedlichen Situationselementwerten besitzen und mittels Pfaden mit dem Normalfall-Situational Scope verknüpft sind. Wenn also der Normalfall-Situational Scope die Elemente X, Y und Z definiert, dann sollte der alternative Situational Scope ebenso X, Y und Z oder eine Untermenge dieser definieren. Untermengen sind erlaubt, da sonst für jede Situation mit n Situationselementen 2 hoch n alternative Situationen definiert werden müssen. Dies bildet jedoch die Realität nicht zwangsläufig ab: Eventuell will man nur einen speziellen Fall abfangen und den Rest ignorieren. Ebenso kann eine *Default*-Situation definiert werden. Diese wird ausgeführt, wenn keine andere Alternativsituation passt.

Nachdem ein Situational Scope ausgeführt worden ist, wird der Kontrollfluss an den nächsten Situational Scope weitergegeben. Für diesen werden die gleichen Schritte, also Evaluation und Ausführung, bzw. bei negativem Ergebnis der Evaluation der Alternativpfad durchgeführt. Dieses Konzept wird nochmals in Abbildung 5.3 dargestellt.

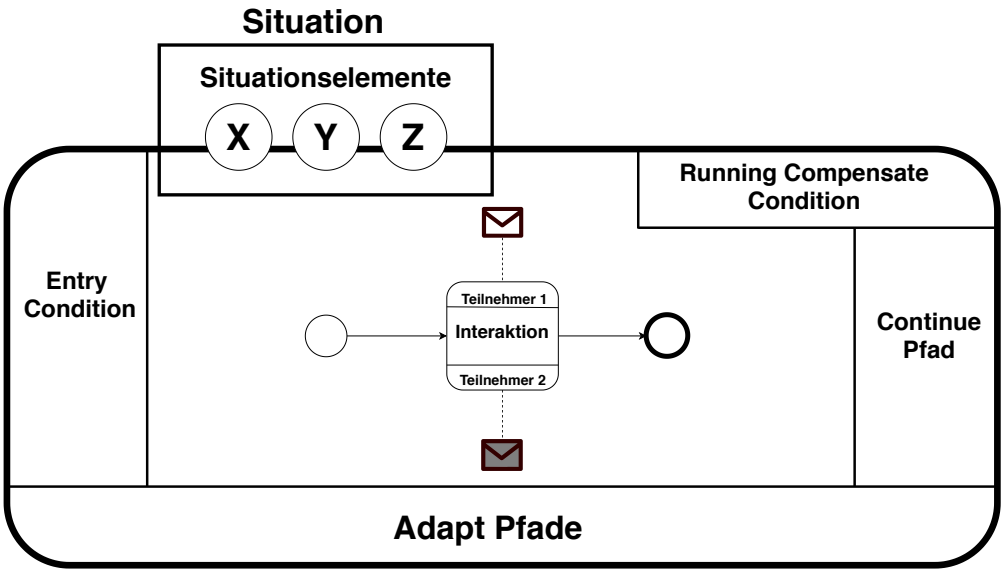


Abbildung 5.2: Darstellung des Situational Scope-Konzepts mitsamt der möglichen Attribute

Im Use Case würde ein Situational Scope beispielsweise die Situation mit dem Namen “Kaffee Kochen“ definieren. Einzelne Situationselemente wären dann “Kaffeemaschine“ und “Kaffeebohnen“, jeweils mit dem Wert “True“ definiert. Zur Laufzeit würden die Werte “Kaffeemaschine“ und “Kaffeebohnen“ evaluiert werden, etwa durch Sensoren. Diese würden beispielsweise prüfen, ob die Kaffeemaschine angeschaltet und verfügbar ist und messen, ob genügend Kaffeebohnen vorhanden sind. Der Situational Scope dürfte nur dann ausgeführt werden, wenn die von den Sensoren gemeldeten Daten zum Ergebnis führen, dass “Kaffeemaschine“ und “Kaffeebohne“ zu “True“ evaluieren. In dem Fall würde der dazugehörige Choreographieprozess gestartet und der Nutzer würde beispielsweise der Kaffeemaschine den Befehl “Koche Kaffee“ schicken. Falls jedoch beispielsweise “Kaffeemaschine“ zu “False“ und “Kaffeebohnen“ zu “True“ evaluieren, wird der Situational Scope mit der Situation “Kaffeemaschine kaputt“ mit passenden “Situationselementen ausgeführt. Dieser macht dann einen Termin mit einem Reparaturservice aus und stößt den “Tee Kochen“ Situational Scope an, welcher eine neue Situation mit den neuen Situationselementen “Teekessel“ und “Teebeutel“ abbildet.

5.1.2 Condition-Definitionen

Entry Conditions und Running Compensate Conditions besitzen verschiedene Attribute die mehrere Werte annehmen können, welche definieren, wie die Evaluation einer Situation und mögliche Abweichungen ausgeführt werden. Die beiden Conditions folgen dabei den Exception Handling Patterns nach Reichert et al. [RW12] (siehe Abschnitt 2.6). Die grundsätzlichen Werte, die für Entry Condition und Running Compensate Condition gesetzt werden können, sind gleich. Die Conditions haben einen *Typ*, welcher definiert, was im Fall, dass die Situation negativ evaluiert wird, geschehen soll. Der Typ kann die folgenden Werte annehmen, die in Abbildung 5.4 dargestellt sind:

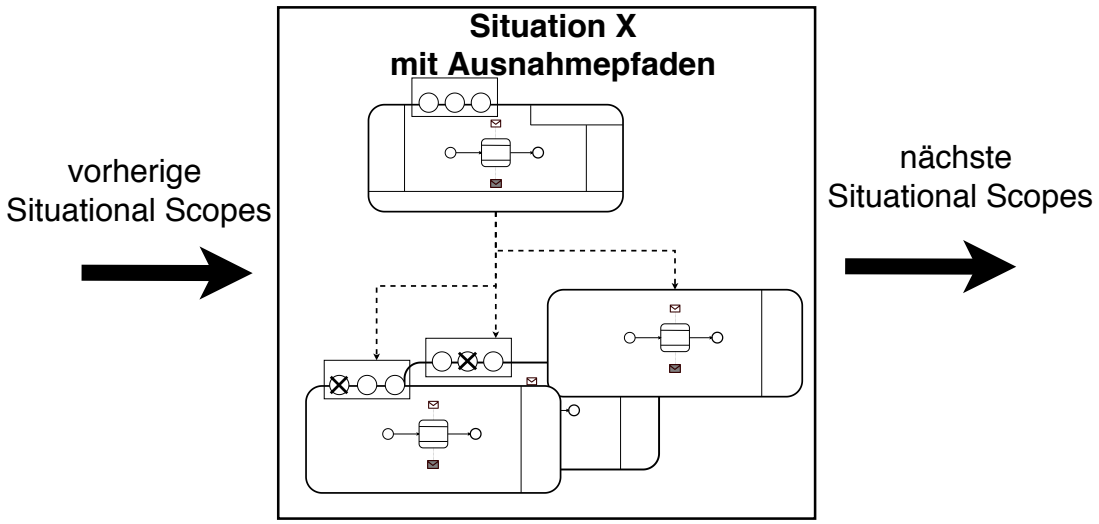


Abbildung 5.3: Darstellung des Zusammenhangs zwischen verschiedenen Situational Scopes. Ein Situational Scope stellt den Standardablauf dar und hat mehrere Situational Scopes mit Alternativabläufen angehängt. Nach Abschluss eines Situational Scopes wird der nächste Situational Scope ausgeführt

- *Adapt*: Evaluiert die angehängten Adapt-Pfade und erstellt entsprechende Exception Flows im Zieldiagramm. Diese Exception Flows definieren die Ausführung der Prozesse, die in den Situational Scopes der Adapt-Pfade modelliert sind.
- *Abort*: Beendet die Ausführung des Pfades und setzt ein End Event an den Exception Flow. Dies bricht den Prozess vollständig ab.
- *Return*: Übergibt die Ausführung mittels Message Event zurück an den Prozess, der im vorhergehenden Situational Scope definiert wurde. Das Message Event wird an ein Message Start Event gekoppelt. Es übergibt also die Kontrolle an die vorhergehende Instanz. Diese muss dann neu entscheiden anhand ihrer Conditions. Dies gibt dem Modellierer die Möglichkeit Zuständigkeiten besser zu modellieren, da es in einem dezentralen Kollaborationsdiagramm keinen Orchestrator hat, der diese Funktion übernimmt.
- *Continue*: Unabhängig davon, ob Situation eintritt oder nicht, wird der Standardpfad ausgeführt. Dieser Wert erzwingt also den Standardpfad, unabhängig vom Evaluationsergebnis. Damit könnten Deferred Fixing-Lösungen umgesetzt werden.
- *Retry*: Retry wiederholt den Prozess.

Ebenso besitzen die Conditions ein *Wait*-Attribut, welches die Werte “True“ oder “False“ annehmen kann. Wenn “Wait“ auf “True“ gesetzt wird, wird ein Timer gesetzt, welcher eine definierbare Zeit darauf wartet, dass die Situation eintritt. Wenn diese abläuft, wird anhand der, in der Condition definierten, Werte gehandelt (siehe Abbildung 5.6).

Für den Fall das ”Wait“ auf “False“ gesetzt wird, wird nur einmalig evaluiert und im negativen Fall direkt der Kontrollfluss anhand der Condition-Werte ausgeführt (siehe Abbildung 5.7).

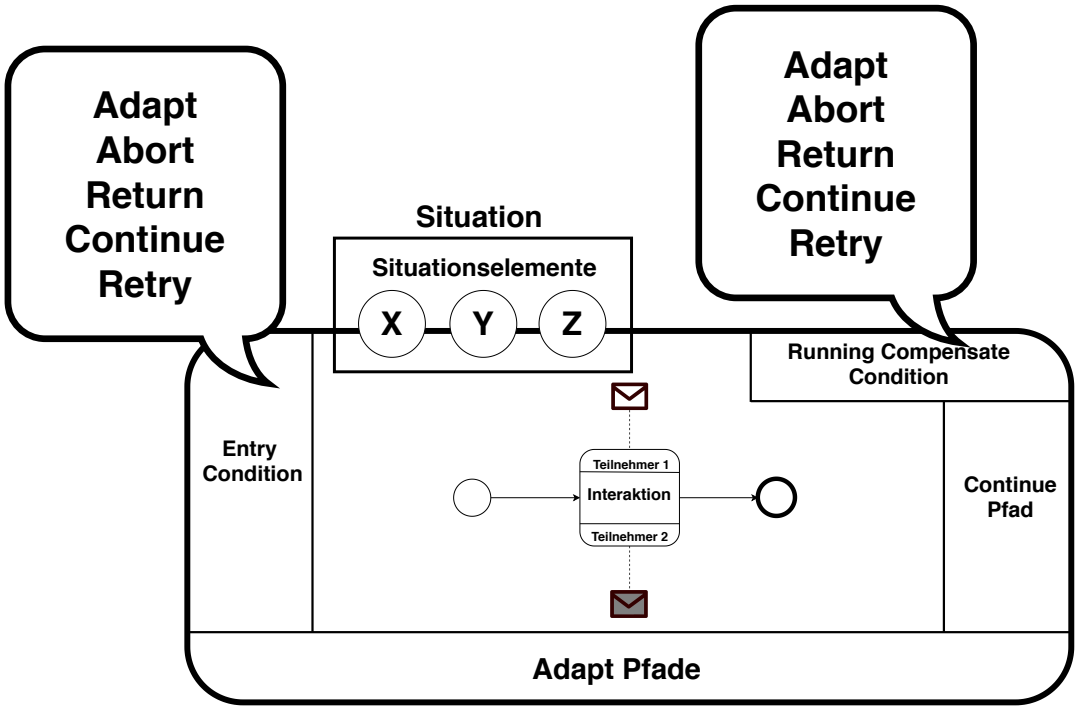


Abbildung 5.4: Werte der Condition-Typen eines Situational Scopes

Die verschiedenen Typen und ihre Übergänge werden nochmals in Abbildung 5.5 als Zustandsdiagramm aufgeführt. Es zeigt die beiden Zustände Evaluation und Ausführung, sowie die Übergänge zwischen den Zuständen, die von den gewählten Condition-Typen abhängig sind. Auch zeigt das Diagramm, welche Endzustände durch die Wahl der Condition-Typen erreicht werden.

Ein weiteres Attribut, welches die Conditions besitzen, ist die *Adaption Strategy*: Diese definiert, welche Adaptionspfade ausgeführt werden sollen. Dieses Attribut kann nur gesetzt werden, wenn als Condition-Typ “Adapt“ gewählt wurde. Die Adaption Strategy kann die Werte *Best fit* oder *All fit* annehmen. Bei “Best fit“ werden nur die Pfade ausgeführt, deren Situationselemente perfekt passen. Bei “All fit“ werden hingegen auch Submengen passender Situationselemente ausgeführt.

Im Use Case könnte man also beispielsweise erneut die ”Kaffee Kochen“-Situation mit den Werten “Kaffeemaschine“ und “Kaffeebohnen“ modellieren. Als alternative Situationen würde man die Situation “Kaffeemaschine kaputt“ mit dem einzigen Situationelement “Kaffeemaschine“ mit Wert “False“ und die Situation “Kaffeemaschine kaputt, aber Bohnen vorhanden“ mit den beiden Situationelementen “Kaffeemaschine==False“ und “Kaffeebohnen==True“ wählen. Im Fall der Situation “Kaffeemaschine kaputt“ würde ein Prozess eine alternative Kaffeemaschine starten und die vorhandenen Kaffeebohnen in diese Kaffeemaschine einfüllen. Der Prozess “Kaffeemaschine kaputt“ würde einen Reparateurtermin ausmachen und dem Nutzer Bescheid geben. Wenn nun bei der Ausführung des Modells bei der Situation “Kaffee kochen“ die Situationselemente zu “Kaffeemaschine==False“ und “Kaffeebohnen==True“ evaluieren, würde bei der Adaption Strategy “Best fit“ nur die alternative Situation “Kaffeemaschine kaputt aber Bohnen vorhanden“ ausgeführt werden, da beide evaluierten Situationselemente auch für die alternative Situation passen müssen. Dies wäre bei der Situation “Kaffeemaschine kaputt“ jedoch nicht der Fall, da hier nur das Element

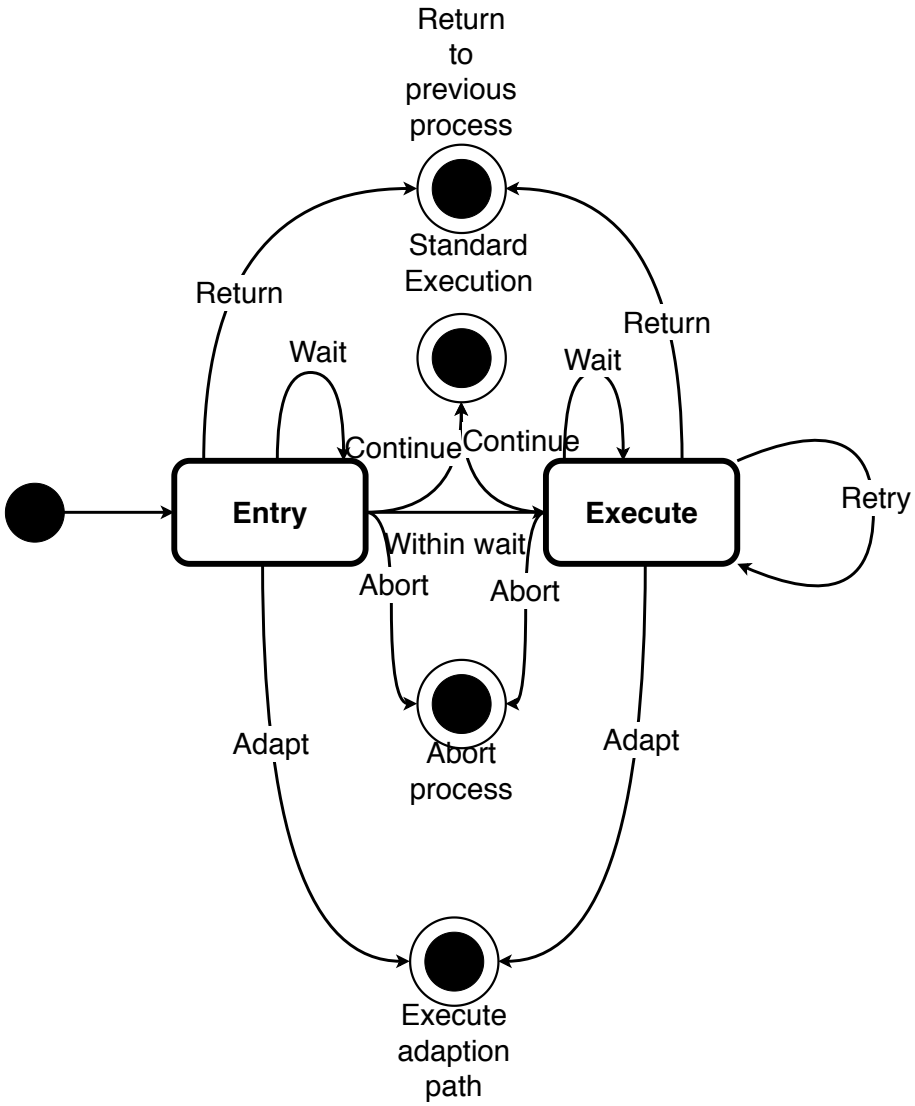


Abbildung 5.5: Diagramm der Zustände des Prozesses und der von den Condition-Typen abhängigen Übergänge

"Kaffeemaschine" übereinstimmt, das andere Element jedoch nicht mal gesetzt wurde. Bei "All Fit" würden beide Situationen ausgeführt werden, da "Kaffeemaschine==False" eine Submenge von "Kaffeemaschine==False" und "Kaffeebohnen==True" ist.

Der Sinn hinter diesem Attribut ist die Möglichkeit, Situationen zu verfeinern. So könnte der Prozessmodellierer Interesse daran haben, dass bestimmte Handlungen für bestimmte Situationselemente immer ausgeführt werden. Anstatt nun für jede Situation, an welcher das Situationselement beteiligt ist, die gleichen Handlungen zu definieren, kann der Modellierer einfach mit "All fit" die Ausführung einer Subsituation für alle möglichen Situationen festlegen. Da mit jedem zusätzlichen Situationselement die Anzahl an möglichen Situationen exponentiell steigt, kann dies, besonders

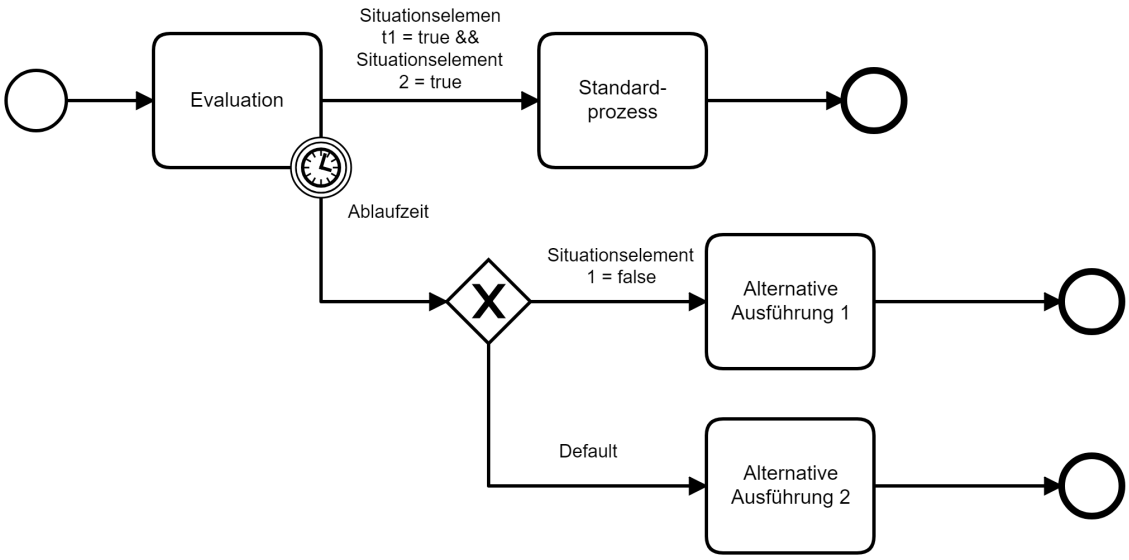


Abbildung 5.6: Umsetzung des Wait-Attributs für den Fall “True“ in einem BPMN-Prozess. Die Wartezeit wird in einem Timer Boundary Event eines speziellen Evaluationsprozesses definiert

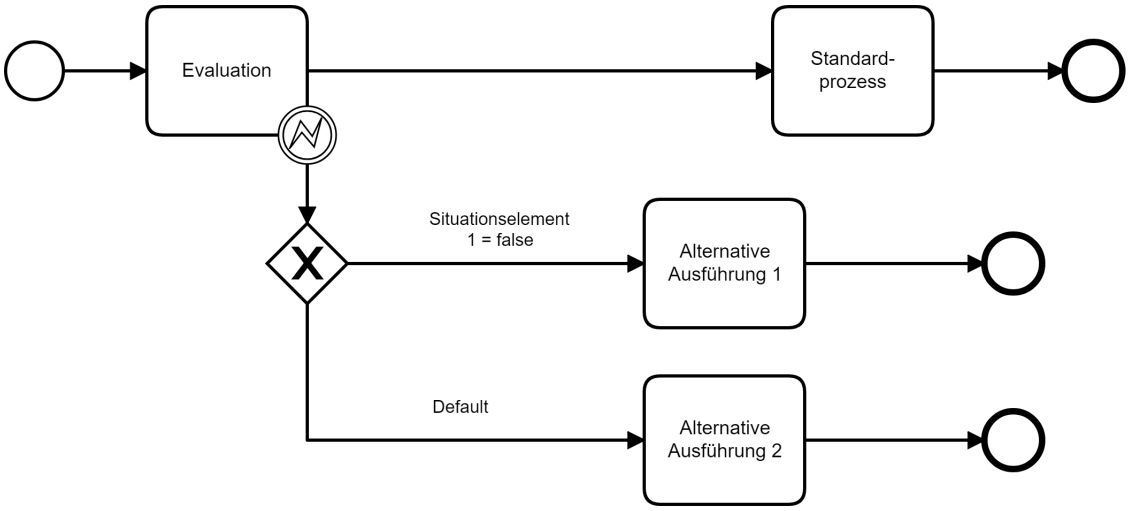


Abbildung 5.7: Umsetzung des Wait-Attributs für den Fall “False“ in einem BPMN-Prozess. Das Vorhandensein einer Situation wird einmalig evaluiert. Im Fall, dass die Evaluation negativ ausfällt, wird ein Error Boundary Event ausgelöst

bei komplexeren Modellen, Modellierarbeit sparen. Im transformierten Kollaborationsdiagramm wird dies umgesetzt mittels XOR-Gateway bzw. Inclusive-Gateway. Das XOR-Gateway erzwingt einen bestimmten Pfad, während das Inclusive-Gateway auch Subpfade zulässt.

Für die Adapt-Pfade kann auch der *Execution-Type* gesetzt werden: Dieser kann die Werte *Interrupting* oder *Non-Interrupting* annehmen. “Non-Interrupting“ führt alle möglichen Pfade zu Ende aus, “Interrupting“ nutzt das Racing Incoming Message Pattern (siehe Abschnitt 3.3), um Prozesse zu

unterbrechen, sobald ein Prozess die Ausführung abgeschlossen hat. Dieses Attribut ermöglicht es mehrere Situationen mit gleichen Situationselementen und gleichen Situationselementwerten zu definieren und festzulegen, wie diese Situationen ausgeführt werden sollen.

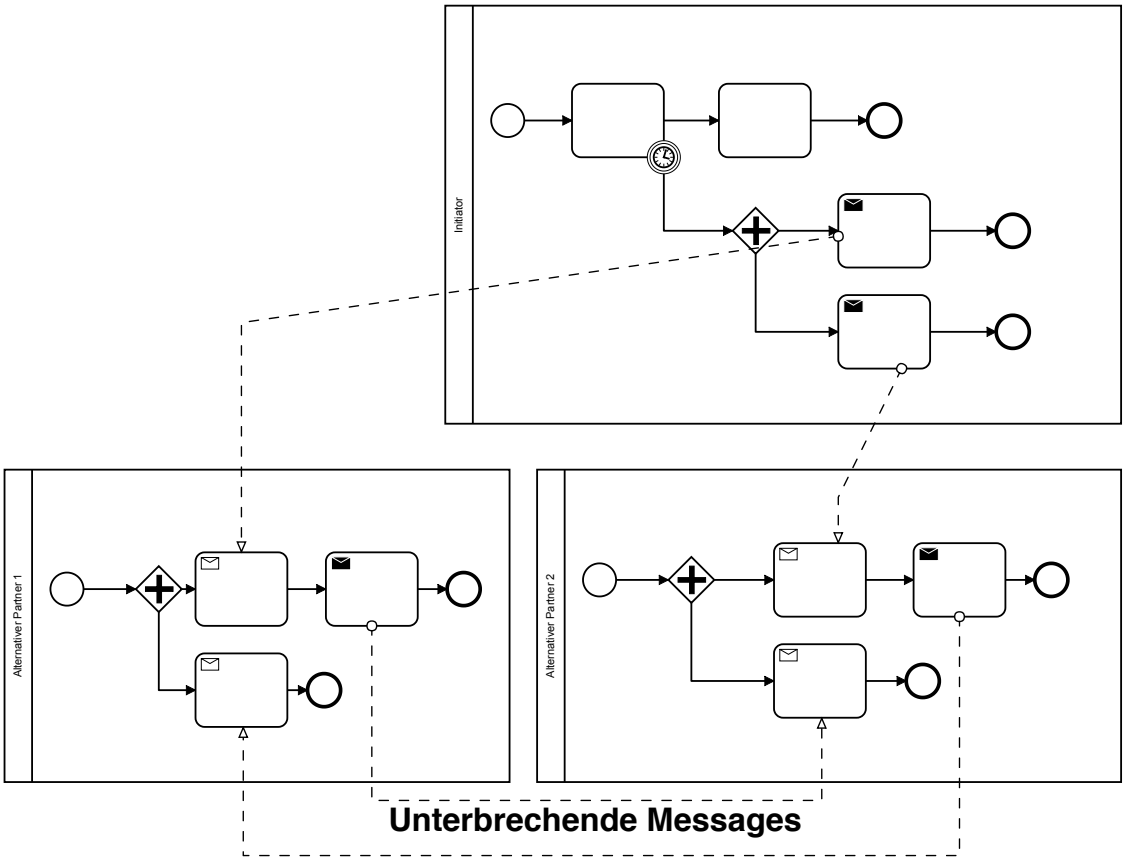


Abbildung 5.8: Darstellung der Umsetzung des "Interrupting"-Werts in einem BPMN-Kollaborationsdiagramm. Hier wird nach Ausführung der technischen Tasks eines Teilnehmerprozesses eine Nachricht an die anderen, parallel ablaufenden, Prozesse gesendet und diese dadurch unterbrochen

Im Use Case könnte man beispielsweise im Falle einer kaputten Kaffeemaschine zwei Alternativsituationen für das Situationselement "Kaffeemaschine" mit dem Wert "False" definieren. Eine Situation würde einen "Tee-Kochen"-Prozess ausführen, die andere Situation einen Kaffee bei einem Lieferdienst bestellen. Mit dem Wert "Non-interrupting" würden beide Prozesse ausgeführt werden, d. h. am Ende hätte der Nutzer sowohl den bestellten Kaffee als auch einen Tee. Mit dem Wert "Interrupting" würde, sobald beispielsweise der "Kaffee-Bestellen"-Prozess abgeschlossen ist, dieser dem "Tee-Kochen"-Prozess eine Nachricht schicken, welche den "Tee-Kochen"-Prozess abbricht. Dies lässt sich auch mit mehr als zwei Situationen modellieren, in dem Fall müsste man das Racing Incoming Message Pattern mit den One-To-Many Send und One-From-Many Receive Pattern kombinieren (siehe Abschnitt 3.3). Die Idee dahinter ist, dass es Prozesse geben kann, in denen es mehrere Subprozesse mit gleichwertigem Ergebnis gibt und es dem Nutzer egal ist, welcher Subprozess denn nun tatsächlich ausgeführt wird, solange nur einer ausgeführt wird. Bei "Non-Interrupting" werden im BPMN-Kollaborationsdiagramm die Prozesse einfach gestartet und

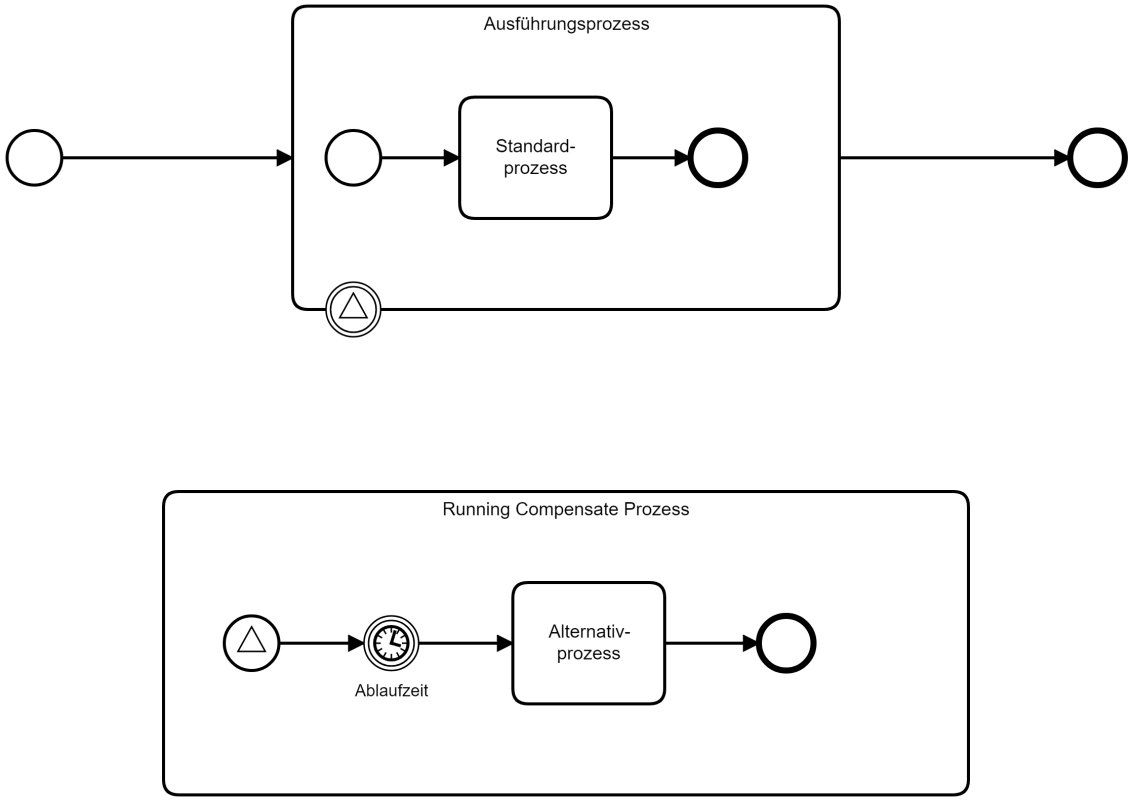


Abbildung 5.9: Umsetzung der Running Compensate Condition in einem BPMN-Prozess. Dafür wird ein Event Sub-Process verwendet, welcher mittels Signal Event gestartet wird, wenn die Situation nicht mehr gültig ist

zu Ende ausgeführt. Bei “Interrupting“ sorgen bei den einzelnen Prozessen nach Ausführung ihrer Tasks, aber vor dem End Event, bei der Transformation platzierte Send Tasks, welche Nachrichten an die Receive Tasks der anderen Prozesse schicken, dafür, dass diese ein End Event erreichen und damit der Prozess beendet wird (siehe Abbildung 5.8).

Entry Condition

Entry Conditions haben noch keine Arbeit ausgeführt, können also direkt eine andere Situation ausführen. Eventuell können zusätzliche Funktionen ausgeführt werden (Adding Function, siehe Abschnitt 2.6). Im Kollaborationsdiagramm werden für Entry Conditions Evaluationsprozesse erstellt, an welchen ein Boundary Event angehängt wird, welcher den Exception Flow im Falle der negativen Situationsevaluation ausführt. Dies ist bereits in den Abbildungen 5.6 und 5.7 abgebildet.

Running Compensate Condition

Bei Running Compensate Condition muss etwas kompensiert werden, da eventuell schon Tasks ausgeführt wurden. Running Compensate Condition wird mittels eines *Event Sub-Process* umgesetzt, welcher im Falle eines Problems mit dem Ausführungssubprozess über ein Event aufgerufen wird.

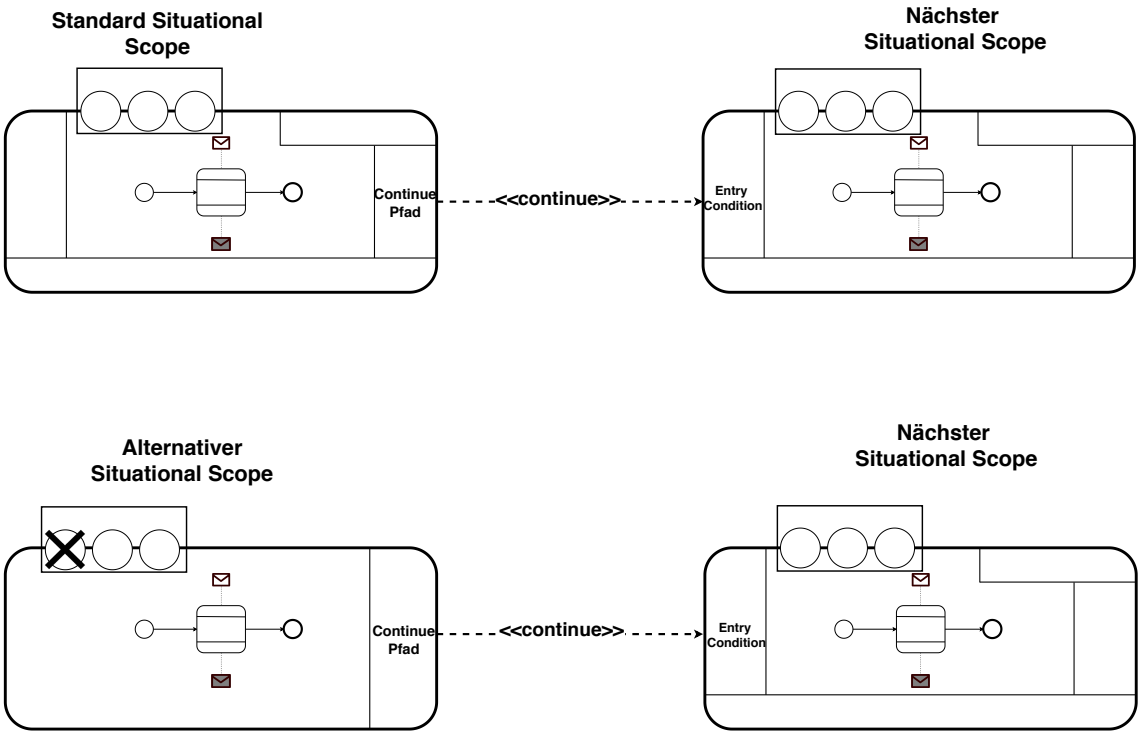


Abbildung 5.10: Continue-Pfade können an Standard-Situational Scopes und an alternative Situational Scopes angehängt werden, um fortlaufende und komplexere Prozesse gewährleisten zu können

Falls die Condition als “Wait==True“ definiert wurde, bleibt dem Hauptprozess die definierte Zeit, um darauf zu warten, dass die Situation wieder gilt. Dies wird mittels Wait Event umgesetzt. Bei “Wait==False“ wird der Event Sub-Process direkt ausgeführt. Hier ähneln die Event Sub-Processes dabei Compensation Spheres [LR00], da sie an einen Situational Scope, also an eine Standardausführung eines Prozesses, angehängt werden. Sie gelten jedoch für mehrere Tasks gleichzeitig, weshalb hier die Einschränkung gemacht werden muss, dass nicht unterschieden werden kann, wie weit der Prozess bereits ausgeführt worden ist. Es muss also eine generelle Kompensation erfolgen. Die Umsetzung als Event Sub-Process ist in Abbildung 5.9 abgebildet.

5.1.3 Adapt- und Continue-Pfade

Adapt- und Continue-Pfade müssen für Entry Condition und für Running Compensate Condition separat definiert werden. Continue-Pfade führt zu neuen Situationen. Adapt-Pfade können als Kompensationspfade angesehen werden, die in eine neue Situation überleiten können müssen. Daher kann an alternative Situational Scopes ein Continue-Pfad angehängt werden, welcher zu einer neuen Situation führt (siehe Abbildung 5.10).

Adapt-Pfade sind also Compensate oder Adding Functions (Siehe Workflow Patterns in Abschnitt 2.6). Adapt-Pfade können Running Compensate- oder Entry-Pfade definieren. Wie die Pfade transformiert werden, hängt von den Eigenschaften der jeweiligen Condition ab, sie können

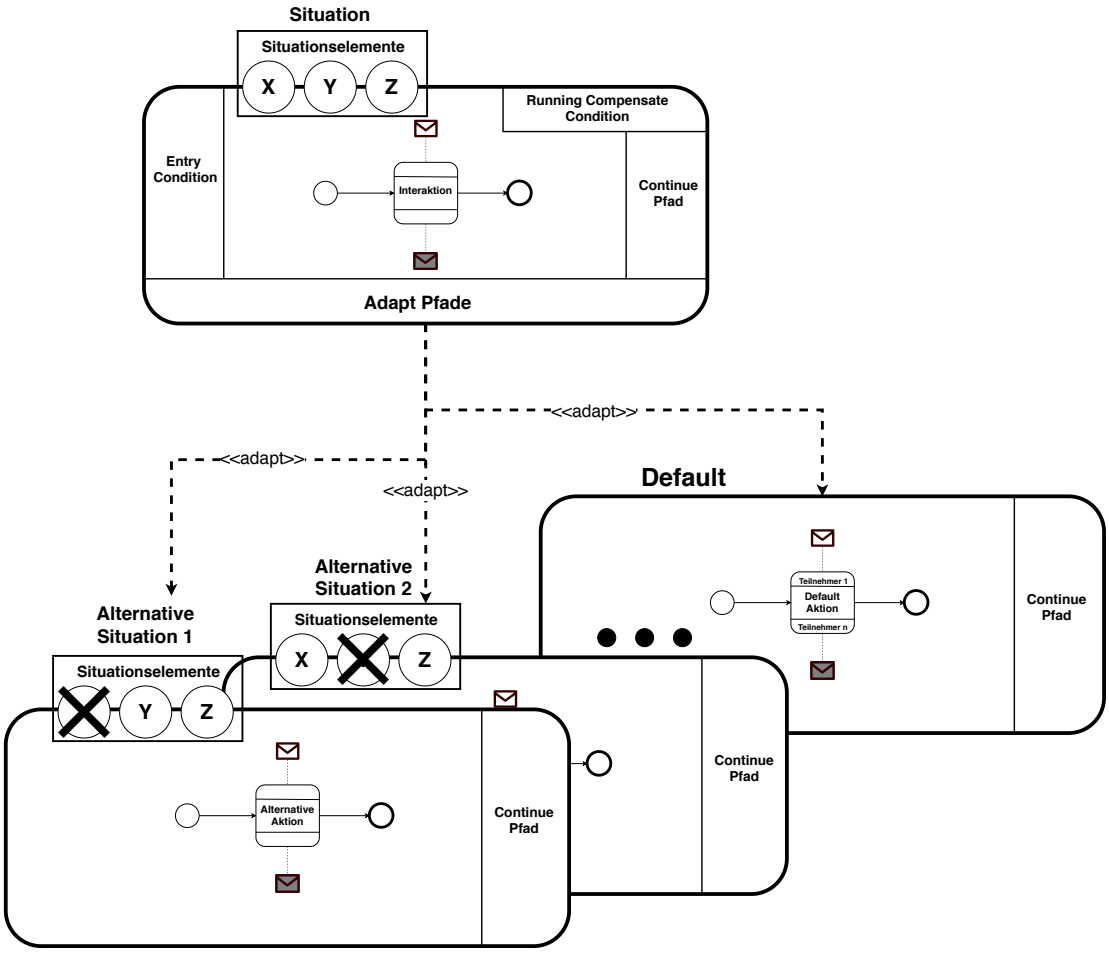


Abbildung 5.11: Adapt-Pfade werden an Situational Scopes angehängt und führen zu alternativen Situational Scopes

also durch XOR- oder Parallel-Gateways verbunden werden. An einen Situational Scope können 0 bis n Adapt-Pfade angehängt werden (siehe Abbildung 5.11). Diese werden zu Exception Flows im Kollaborationsdiagramm.

5.2 Transformation von Choreographie zu Kollaboration

Für die Transformation von Choreographie- zu Kollaborationsdiagramm wurde im Rahmen dieser Thesis ein Algorithmus entwickelt. Dabei wurden die folgenden, bereits im Abschnitt 3.2 vorgestellten Überlegungen von Bischoff et al. [BFR19] beachtet: So muss die Struktur der Choreographie in der Kollaboration erhalten bleiben. Beispielsweise müssen Gateways übereinstimmen und einen entsprechenden Kontrollfluss in mehreren Teilnehmern definieren. Allerdings sollten auch keine Gateways existieren, wenn der Kontrollfluss an einer Stelle nicht gesteuert werden muss. Unnötige Strukturen sollten also nicht übernommen werden.

Die Transformation wurde in zwei Algorithmen unterteilt. In beiden Algorithmen wird das Choreographiediagramm mittels Tiefensuche durchiteriert. Der erste Algorithmus bereitet die Transformation vor, indem er, für die Transformation notwendige, Datenstrukturen erstellt und ist in Algorithmus 5.1 abgebildet. Dazu zählt er die Anzahl der Choreography Tasks an denen die einzelnen Teilnehmer beteiligt sind und speichert die besuchten Elemente pro Teilnehmer (Siehe Zeilen 18-27). Dabei werden Elemente pro Pfad erst gespeichert, wenn der erste Choreography Task gefunden wurde, da der Teilnehmer als Schlüssel in der Map verwendet wird (Siehe Zeilen 30-32) und erst angelegt wird, sobald der erste Choreography Task gefunden wurde (Siehe Zeile 25). Dadurch werden die für den jeweiligen Teilnehmer unnötigen Elemente am Anfang des Diagramms abgeschnitten. Die Anzahl der Elemente wird gespeichert, um im zweiten Algorithmus überprüfen zu können, wie viele Choreography Tasks bereits besucht und transformiert worden sind. Wenn alle bereits transformiert worden sind, müssen keine weiteren Elemente hinzugefügt werden. Dies schneidet die für den jeweiligen Teilnehmer unnötigen Elemente am Ende des Diagramms ab. Die durch den Algorithmus gesammelten Informationen werden an den nächsten Algorithmus übergeben, welcher mittels dieser Informationen das Choreographiediagramm in ein Kollaborationsdiagramm transformiert (Siehe Zeile 39).

Algorithmus 5.1 Zähle Choreography Tasks der Teilnehmer und finde Elemente

```
1:  //Choreographiediagramm D und Starteventknoten s
2:  procedure BEREITETRANSFORMATIONVOR(D,s)
3:      //Mappt Teilnehmer zu Anzahl an Choreography Tasks pro Teilnehmer
4:      Map<ID,Integer> tasksProTeilnehmer
5:      //Mappt Teilnehmer zu Elementen die an Kollaboration des Teilnehmers beteiligt sind
6:      Map<ID,List<Task>> teilnehmerElemente
7:      //Stack für Tiefensuche
8:      Stack S
9:      S.push(s)
10:     while S nicht leer do
11:         v=S.pop()
12:         //Durchsuche die Folgeelemente des Elements v
13:         for all Folgeelemente w von v in Diagramm D do
14:             if w ist nicht besucht then
15:                 //Wenn w ein Choreography Task ist, muss er gezählt und zu den Elementen
                    des Teilnehmers hinzugefügt werden
16:                 if w ist Choreography Task then
17:                     //Ein Choreography Task hat eine Liste der Teilnehmer. Für diese Teilneh-
                        mer wird der Task hinzugefügt und gezählt
18:                     for Teilnehmer p in Teilnehmer von w do
19:                         //Fügt das Element w zu der Liste der für den Teilnehmer relevanten
                            Elemente hinzu
20:                         teilnehmerElemente[p].add(w)
21:                         //Wenn p bisher noch an keinen Tasks beteiligt war, hat er keinen
                            Schlüssel in tasksProTeilnehmer. Dann muss er erst hinzugefügt wer-
                            den. Dies verhindert, dass andere Elemente in die Liste der relevanten
                            Elemente geschrieben werden
22:                         if p in Schlüssel von tasksProTeilnehmer then
23:                             tasksProTeilnehmer[p]=tasksProTeilnehmer[p]+1
24:                         else
25:                             tasksProTeilnehmer[p]=1
26:                         end if
27:                     end for
28:                 else
```

Algorithmus 5.1 Zähle Choreography Tasks der Teilnehmer und finde Elemente (Fortsetzung)

```
29:                     //Andere Elemente wie Events und Gateways werden in der Liste der
                        bereits bekannten Teilnehmer gespeichert. Dies verhindert, dass unnötige
                        Elemente am Anfang in die Liste geschrieben werden
30:                     for n in Schlüssel von tasksProTeilnehmer do
31:                         tasksProTeilnehmer[n].push(w)
32:                     end for
33:                 end if
34:             end if
35:             S.push(w)
36:             Markiere w als besucht
37:         end for
38:     end while
39:     return tasksProTeilnehmer, teilnehmerElemente
40: end procedure
```

Der zweite Algorithmus ist in Algorithmus 5.2 dargestellt und nimmt die Daten des ersten Algorithmus (Siehe Zeile 3) und baut pro Teilnehmer den Prozess auf. Er speichert in einer Map ab, welche Choreography Tasks zu welchen Receive und Send Tasks transformiert wurden (Siehe Zeile 5 und Zeilen 27 + 30). Für die Transformation nutzt er die Anzahl der Choreography Tasks pro Teilnehmer um zu prüfen, wann keine neuen Elemente hinzugefügt werden müssen (Siehe Zeile 22). Wenn die Anzahl der besuchten Choreography Tasks (momentaneAnzahlTasks, definiert in Zeile 13) gleich der Anzahl der maximal besuchbaren Choreography Tasks (anzahlTasks, definiert in Zeile 11) ist, müssen keine neuen Send oder Receive Tasks hinzugefügt werden. Auch Gateways müssen nicht mehr gesetzt werden, da in der Tiefensuche nie der momentane Graphknoten, also das momentane Element, transformiert wird, sondern immer die Kindknoten (Siehe Zeilen 19-46). Es werden also immer automatisch die Gateways für einen Pfad geschlossen. Da das Diagramm mittels Tiefensuche durchsucht wird und deshalb bereits besuchte Elemente nicht nochmals betrachtet werden, werden Sequence Flow-Verbindungen zwischen aktuellem Element und nächstem Element nicht erstellt. Daher muss dieser Fall abgefangen werden und entsprechende Verbindungen erstellt werden. Dazu sucht er das im Choreographiediagramm nächste bereits besuchte Element, holt sich das dazugehörige Element im Kollaborationsdiagramm und verbindet das momentane Element mit diesem (Siehe Zeilen 39-41). Nachdem alle Strukturen aufgebaut sind, wird die Map von Choreography Tasks zu Receive/Send Tasks dann verwendet, um Message Flows zwischen diesen aufzubauen (Siehe Zeilen 50-52). Unnötige Elemente am Anfang und am Ende sind nun nicht transformiert worden, jedoch können, je nach Diagramm, zwischendrin noch redundante Gateways existieren. Diese werden nochmals durchgegangen und gelöscht, nämlich genau dann, wenn ein Pfad von Gateways umgeben ist, welche nur mit einem Pfad verbunden sind (Siehe Zeilen 54-60).

Algorithmus 5.2 Transformiere das Choreographiediagramm in ein Kollaborationsdiagramm

```
1: //Choreographiediagramm D und Starteventknoten s
2: procedure TRANSFORMIEREDIAGRAMM(G,s)
3:     tasksProTeilnehmer, teilnehmerElemente= BEREITETRANSFORMATIONVOR(D,s)
4:     Kollaborationsdiagramm K
5:     Map<ChoreographyTaskID, (SendTaskID,ReceiveTaskID)> choreographyTaskMapping
6:     //Iteriert durch alle Teilnehmer des Choreographiediagramms
7:     for Teilnehmer p in D do
8:         teilnehmerProzess = erstelle neuen Teilnehmerprozess in K
9:         //Holt sich die Anzahl der Tasks des Teilnehmers und die Liste der für diesen
           Teilnehmer relevanten Elemente
10:        momentaneTeilnehmerElemente=teilnehmerElemente[p]
11:        anzahlTasks=tasksProTeilnehmer[p]
12:        //Zählt die Anzahl der besuchten Elemente
13:        momentaneAnzahlTasks=0
14:        Stack S
15:        S.push(s)
16:        Markiere s als besucht
17:        while S ist nicht leer do
18:            v= S.pop()
19:            for all Nachbarn w von v in Diagramm D do
20:                if w ist nicht besucht then
21:                    if w in momentaneTeilnehmerElemente then
22:                        if anzahlTasks>momentaneAnzahlTasks then
23:                            if w ist Choreography Task then
24:                                //w hat ein Attribut, welches den initiierenden Teilnehmer an-
                                   gibt. Der Block hängt den erstellten Task an den Prozess an
                                   und mappt ihn zum Choreography Task
25:                                if (Initiierender Teilnehmer von w)==p then
26:                                    st=füge neuen sendTask zu teilnehmerProzess hinzu
27:                                    choreographyTaskMapping[w].push(st)
28:                                else
29:                                    rt= füge neuen Receive Task zu teilnehmerProzess hinzu
30:                                    choreographyTaskMapping[w].push(rt)
31:                                end if
32:                                momentaneAnzahlTasks=momentaneAnzahlTasks+1
33:                            else
34:                                //Wenn w kein Choreography Task ist, ist er ein Gateway oder
                                   Event. Das entsprechende Element muss an den Prozess ange-
                                   hängt werden
35:                                Füge Element des Typs w zu teilnehmerProzess hinzu
36:                            end if
37:                        end if
```

Algorithmus 5.2 Transformiere das Choreographiediagramm in ein Kollaborationsdiagramm (Fortsetzung)

```
38:                         //Wenn das nächste Element von w im Ursprungsdiagramm bereits
                            besucht wurde und ein relevantes Element ist, gibt es einen Sequence
                            Flow zwischen diesen beiden Elementen. Da die Tiefensuche dieses
                            Element nicht mehr besuchen würde, muss dieser Sequence Flow an
                            dieser Stelle auch im neuen Prozess erstellt werden
39:                         if Nächstes Element n von w ist besucht then
40:                             Erstelle Sequence Flow zwischen Element w und nächstem
                                Element n
41:                         end if
42:                     end if
43:                     S.push(w)
44:                     Markiere w als besucht
45:                 end if
46:             end for
47:         end while
48:     end for
49:     //Mittels der Map zwischen Choreography Tasks und Send/Receive Tasks werden die
        Message Flows erstellt
50:     for Elemente e in choreographyTaskMapping do
51:         Verbinde Send Task von e mit Receive Task von e mit Message Flow
52:     end for
53:     //Wenn ein Gateway nur einen leeren Pfad hat ist er unnötig und kann gelöscht werden.
        Dies ist der Fall, wenn beide Elemente a und b Gateways und beide nur einen ausge-
        henden Pfad haben. Das erste Gateway ist dann ein Split-Gateway und das zweite ein
        Merge-Gateway
54:     for all Paare von aufeinanderfolgenden Elementen p(a,b) in Kollaborationsdiagramm K do
55:         if a ist Gateway und b ist Gateway und Anzahl der ausgehenden
56:         Sequence Flows von a == 1 und Anzahl der ausgehenden
57:         Sequence Flows von b == 1 then
58:             Lösche a und b
59:         end if
60:     end for
61: end procedure
```

5.3 Anreicherung mit technischen Prozessen und Ausführung

Choreographiediagramme stellen nur die Interaktionen zwischen den verschiedenen Teilnehmern dar. Da das Modellierungskonzept keine Möglichkeiten vorsieht auch technische Prozesse in situationsbasierten Choreographiemodell zu definieren, fehlen diese auch im transformierten Kollaborationsdiagramm. Daher muss, um das Kollaborationsdiagramm auch ausführen zu können, dieses mit den, lokal für die einzelnen Teilnehmer relevanten, Informationen angereichert werden. Dazu müssen Modellierer Tasks, die die interne Logik beschreiben, in die lokalen Prozesse der

Teilnehmer integrieren, indem sie die Tasks, entsprechend des lokalen Prozessablaufs, zwischen die Interaktions-Tasks platzieren und mit diesen verbinden. Da es sich hierbei um normale und bekannte Modellierung handelt, wird der Vorgang in dieser Thesis nicht weiter erläutert.

Das so entstandene Modell wird auf Workflow-Engines ausgerollt. Abhängig von der Systemarchitektur kann das Modell auf einer oder mehreren Workflow-Engine(s) ausgeführt werden. Da es sich hierbei auch um einen etablierten Vorgang handelt, wird auch dieser nicht weiter erläutert.

6 Implementierung

In diesem Kapitel wird die Umsetzung des Konzepts als Prototyp erläutert. Dafür werden die verwendeten Technologien vorgestellt und anhand von Beispielen die Implementierung dargestellt.

6.1 Verwendete Technologien

Zunächst werden die verwendeten Technologien erklärt. Weitere Informationen zu den Technologien können auf den referenzierten Webseiten gefunden werden.

6.1.1 Camunda Modeler

Camunda ist eine Firma, welche sich auf das Business Process Management (BPM) spezialisiert hat. Seit 2013 entwickeln sie mit Camunda BPM eine auf Java basierende Workflow-Plattform, welche das *BPMN.io*-Toolkit nutzt. Dieses Toolkit implementiert dabei den BPMN 2.0 Standard [Mey13]. Diese Plattform umfasst Werkzeuge für das Modellieren, Überwachen, sowie das Ausführen vom BPMN-Modellen an. Das Modellierungswerkzeug wird *Camunda Modeler* genannt. Der Modeler wird auch als eigenständige Desktop-Anwendung angeboten und unter Verwendung zahlreicher Technologien, wie beispielsweise *Javascript* und *Node.js*, entwickelt [Gmb20c]. Ebenso ist der Modeler Open Source und auf Erweiterbarkeit ausgelegt. Zu diesem Zweck verfügt er über einen eingebauten Plugin-Mechanismus. Der Code wird unter der MIT-Lizenz zur Verfügung gestellt [Gmb20d]. Es existiert viel Dokumentation für Anwendung und Entwicklung des Modelers, welche auch von der Community gepflegt und stetig erweitert wird [Gmb20e; Gmb20f].

In Abbildung 6.1 ist die Benutzeroberfläche des Camunda Modelers abgebildet. Sie besteht aus drei Hauptkomponenten: Der Menüleiste oben, dem Modellierungsbereich in der Mitte und der Log-Konsole unten. Die Menüleiste stellt viele Funktionen zur Interaktion mit dem gesamten Diagramm zur Verfügung, wie etwa das Speichern, das Laden oder der Export als Bild-Datei. Die Log-Konsole stellt Informationen im Fehlerfall zur Verfügung, zum Beispiel wenn ein Diagramm importiert wird, welches nicht den BPMN-Definitionen entspricht. Der Modellierungsbereich besteht wiederum aus mehreren Unterelementen. Auf der linken Seite befindet sich die sogenannte “Palette“, welche sämtlichen verfügbaren BPMN-Elemente enthält. In der Mitte ist der Editor selbst, auf welcher das Diagramm erstellt und bearbeitet wird. Sowohl mit der Palette als auch den Elementen des Editors wird mittels “Drag-and-Drop“ interagiert. Elemente lassen sich jedoch nur platzieren, wenn dies die BPMN-Definition zulässt. So lassen sich zum Beispiel keine Pools innerhalb vorhandener Pools platzieren. Eine weitere Funktion des Editors ist das “Context-Pad“. Dieses öffnet sich, sobald man auf ein Element des Diagramms klickt. Es zeigt sämtliche Elemente an, welche an das ausgewählte Element gemäß BPMN-Definition angehängt werden können. Mit einem Klick auf eines dieser Elemente wird dieses auch automatisch platziert und an das ausgewählte Element

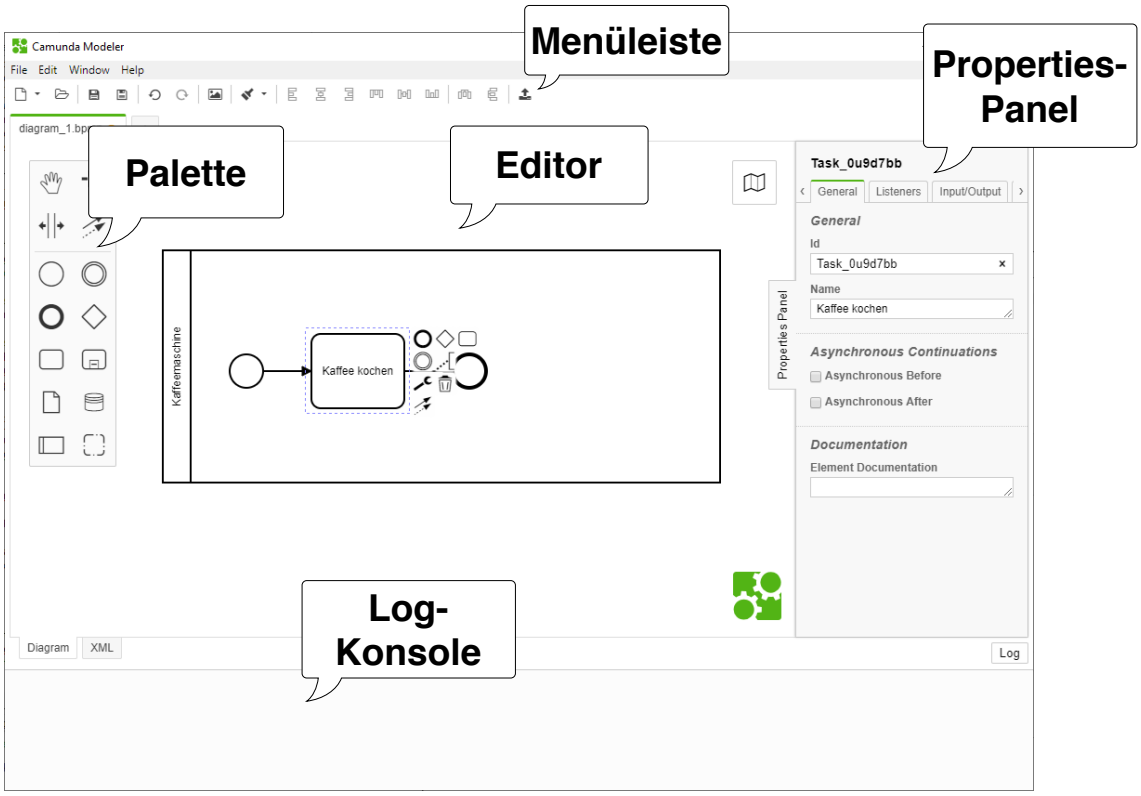


Abbildung 6.1: Benutzeroberfläche des Camunda Modelers

angehängt. Das letzte Element des Modellierungsbereichs ist das Properties-Panel. Dieses zeigt sämtliche vorhandenen und möglichen Eigenschaften des im Editor ausgewählten Elements an. So kann man beispielsweise Ein- und Ausgabeparameter für Tasks definieren.

Da der Camunda Modeler Open Source ist, über einen Plugin-Mechanismus verfügt und viel Dokumentation dazu existiert, bietet er sich gut als Grundlage für die Prototyp-Entwicklung an.

6.1.2 Electron und Node.js

Der Camunda Modeler nutzt das *Electron*-Framework. Dieses Framework vereinfacht die Entwicklung von Desktop-Anwendungen, die auf verschiedenen Plattformen genutzt werden können [Inc20a]. Dies erreicht es, indem es den Webbrowser *Chromium* als Grundlage für eine Anwendung nutzt. Darauf kann der Entwickler mithilfe von Mitteln der Webentwicklung, also Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und Javascript, seine Anwendung implementieren. Da Chromium plattform-übergreifend funktioniert, kann entsprechend der gesamte Code gebündelt werden und auf verschiedenen Betriebssystemen ausgeführt werden, wodurch ein großer Teil der Portierungskosten gespart werden. Auch kann man damit die Anwendung einfacher als Online-Service anbieten, da große Teile des Codes in eine Webseite eingebunden werden können.

Für Electron wird Node.js verwendet. Node.js ist eine Plattform für die server-seitige Ausführung von Javascript-Code [Fou20]. Der Code muss also nicht im Webbrowser selbst ausgeführt werden, sondern kann auf andere Ressourcen zugreifen. Dies erreicht es mit der Einbindung von *Modules*.

Diese sind einzelne Dateien, welche als Module deklariert werden und bestimmte Funktionen erfüllen. Auf diese Weise können Modules nicht nur hinzugefügt, sondern auch überschrieben werden. Der Camunda Modeler nutzt dieses Module-System, um die Einbindung von Plugins und damit die Anpassung des Modelers an die Bedürfnisse des Entwicklers zu ermöglichen [Inc20b].

6.1.3 Chor-js

Ein Nachteil des Camunda Modelers ist die mangelnde Unterstützung von Choreographiediagrammen. Dieses Problem lässt sich jedoch mithilfe des *chor-js*-Frameworks umgehen [LWW19]. Dieses setzt die in BPMN.io definierten Choreographiediagramme als Editor um. Das chor-js-Framework wird vom Hasso-Plattner-Institut implementiert. Da BPMN.io als gemeinsame Grundlage fungiert, bietet es sich an, um Choreographiediagramme im Camunda Modeler umzusetzen. Dabei ist es strukturell gleich zum *bpmn-js*-Framework, welches für die Implementierung des Editors für die Kollaborations- und Orchestrationsdiagramme im Camunda Modeler genutzt wird. Es funktioniert analog zu bpmn-js und lässt sich mit geringem Aufwand im Camunda Modeler einbauen.

6.2 Umsetzung

Das Konzept wurde im Prototyp mittels zweier Plugins umgesetzt: Ein Plugin für die Modellierung des Konzepts und ein Plugin für die Transformation des Models in Standard-BPMN-Kollaborationsdiagramme. Leider ließ sich das Konzept nicht in einem Plugin umsetzen, da sowohl im Camunda Modeler als auch in chor-js noch nicht alle Modules per Plugin-Mechanismus abgeändert werden können. Darum musste für die Einbindung von chor-js im Camunda Modeler im Quellcode des Modelers selbst Code abgeändert werden. Es sind jedoch nur kleine Änderungen von wenigen Zeilen und da sowohl Camunda Modeler als auch chor-js weiterentwickelt werden und auch laut Aussage der Entwickler der Plugin-Mechanismus um die fehlenden Elemente erweitert wird, wird man in Zukunft beide Plugins zu einem zusammenfügen können.

Das Modellierungs-Plugin verwendet chor-js und erweitert die Palette und das Context-Pad des Modelers um die im Konzept vorgestellten Elemente. Da der Editor das willkürliche Setzen von Elementen und Attributen nicht zulässt, sondern nur solcher wie in den Meta-Modell-Definitionen beschrieben, musste das Meta-Modell des Konzepts im Modeler eingebunden werden. Meta-Modelle werden hierbei als JSON-Dateien definiert. Der Code in Listing 6.1 zeigt anhand der Situationselemente einen Ausschnitt, wie die Definition im Plugin umgesetzt wurde. Auf diese Weise sind auch die Meta-Modelle der Kollaborations- und Orchestrationsdiagramme in bpmn-js, als auch die Choreographiediagramme in chor-js eingebunden. Hierbei kommt das bpmn-moddle-Module zum Tragen [Gmb20b]. Dieses definiert das BPMN 2.0-Metamodell und ist für das Schreiben und Lesen von BPMN 2.0-Diagrammen zuständig. Es bietet auch einen Erweiterungsmechanismus, mit welchem eigene Definitionen registriert werden können. Dafür müssen das entsprechende Modul und die Definition geladen werden. Mithilfe der passenden Funktion des Moduls wird die Definition dann eingebunden (siehe Code in Listing 6.2). Damit wird dem Editor die Definition übergeben, welche Elemente und Attribute gesetzt werden dürfen.

Listing 6.1 Ausschnitt aus der JSON-Datei, die das Situational Scope-Konzept definiert

```
...
{
  "name": "SituationElement",
  "superClass": [
    "bpmn:BaseElement"
  ],
  "properties": [
    {
      "name": "situationname",
      "type": "String",
      "isAttr": true
    },
    {
      "name": "situationtrigger",
      "type": "Boolean",
      "isAttr": true
    }
  ]
}
...
```

Listing 6.2 Registrierung der JSON-Datei, die das Situational Scope-Konzept definiert

```
  ...
  import { registerBpmnJSModdleExtension } from 'camunda-modeler-plugin-helpers';
  import ModdleExtension from '../resources/sitscope.json';
  registerBpmnJSModdleExtension(ModdleExtension);

  ...
```

Als Nächstes musste das tatsächliche Setzen von Elementen und Attributen ermöglicht werden. Hierzu wurde die Funktion des Context-Pads und der Palette erweitert. Wie das gemacht wurde, wird hier am Beispiel des Context-Pads erklärt. Zunächst wird dazu der dafür zuständige Provider geladen. Ein Provider ist hierbei ein Modul, welches die Funktionen einer bestimmten zusammenhängenden Funktionsgruppe bietet. Beispielsweise werden die Funktionen des Context-Pads über den “Context-PadProvider“ aufgerufen. Provider können, wie alle anderen Modules, mittels Bezeichnern geladen oder durch das Aufrufen des entsprechenden Objekts über das Elternobjekt. Im Codeausschnitt in Listing 6.3, welches die relevanten Codestellen für das Erweitern des Context-Pads darstellt, wird der Weg über das Elternobjekt aufgezeigt. Hier wird das Module Context-Pad, über seinen Bezeichner namens “contextPad“ geladen. Dann wird eine neue Klasse “CustomContextPad“ erstellt, welche die benötigten erweiterten Funktionen implementiert. Diese Klasse wird dann als Provider beim Elternobjekt “contextPad“ registriert und beim Laden des Context-Pads durch den Modeler aufgerufen. Die Funktion “appendSituationScope“ implementiert alle Schritte, welche beim Erstellen eines neuen Situational Scopes durchgeführt werden müssen. Zunächst muss das Objekt erstellt und sämtliche Standardattribute gesetzt werden. Daraufhin muss auch noch das visuell dargestellte

Diagrammobjekt erstellt und mit dem Modellobjekt verknüpft werden. Der “return“-Abschnitt definiert schließlich, wie das “append.sit-scope“-Element im Context-Pad angezeigt wird und dass beim Klicken auf das Element die Funktion “appendSituationScope“ ausgeführt werden soll. Die Darstellung wird mittels CSS- und HTML-Dateien definiert. Die gesamte Klasse wird als “export default“ deklariert, wodurch das Module für andere Modules sichtbar wird und von diesen auch importiert werden kann. Für die Darstellung der im Editor platzierten Elemente muss auch der “Renderer“ überschrieben werden. Dies wird jedoch analog zu den Providern gemacht.

Listing 6.3 Erweitern des Context-Pad-Providers

```
  CustomContextPad.$inject = [
  'bpmnFactory',
  'config',
  'contextPad',
  'create',
  'elementFactory',
  'injector',
  'translate',
  'moddle'
  ];

  ...

  export default class CustomContextPad {
  constructor(bpmnFactory, config, contextPad, create, elementFactory, injector, translate,
moddle) {
  ...
  contextPad.registerProvider(this);
  ...

  function appendSituationScope() {
    return function(event) {
      const businessObject = bpmnFactory.create('bpmn:SituationScope');
      businessObject.entryCondition="Abort";
      businessObject.waitforentry="false";
  ...

      const shape = elementFactory.createShape({
        type: 'bpmn:SituationScope',
        isExpanded: true,
        businessObject: businessObject
      });
      create.start(event, shape);
    }
  }
    ...
    return {
      'append.sit-scope': {
        group: 'model',
        className: 'bpmn-icon-sitscope',
        title: translate('Append Situational Scope'),
        action: {
          click: appendSituationScope(),
        }
      }
    };
```

Da nicht alle Konzeptelemente visuell dargestellt werden, muss auch das Properties-Panel erweitert werden. Dieses verfügt ebenso über einen Provider. Allerdings müssen hier sogenannte “Tabs“ und “Groups“ hinzugefügt werden. “Tabs“ stellen dabei einzelne Seiten im Properties-Panel dar und “Groups“ Unterabschnitte innerhalb dieser. Im Codeausschnitt in Listing 6.4 werden die benötigten Codezeilen vorgestellt. Innerhalb der Funktion “createSituationScopeTab“ werden zunächst das Label und die ID für die Gruppe für die Situational Scope-Attribute definiert. Diese Information wird dann als Variable an die vorher importierte “createSituationScopeGroup“-Funktion übergeben. Ein Teil dieser Funktion ist im Codeausschnitt in Listing 6.5 dargestellt. Diese Funktion definiert, hier am Beispiel der Entry Condition-Attribute, die einzelnen Attribute der Gruppe. Als Werte der Attribute sind hier nur solche erlaubt, wie sie im Meta-Modell definiert worden sind. Um dies sicherzustellen, muss man die im Meta-Modell definierte “modelProperty“ angeben. Das Ergebnis der Funktion wird an die “createSituationScopeTab“-Funktion zurückgegeben, die wiederum im “CustomPropertiesProvider“ aufgerufen wird. Hier werden die “Tabs“ analog zu den “Groups“ zum kompletten “PropertiesProvider“ kombiniert, welcher vom Properties-Panel genutzt wird. Das fertige Modellierungstool ist in Abbildung 6.2 dargestellt. Es erlaubt das Platzieren von Situational Scopes und markiert die einzelnen Situationselemente, sowie die verschiedenen Pfade farblich.

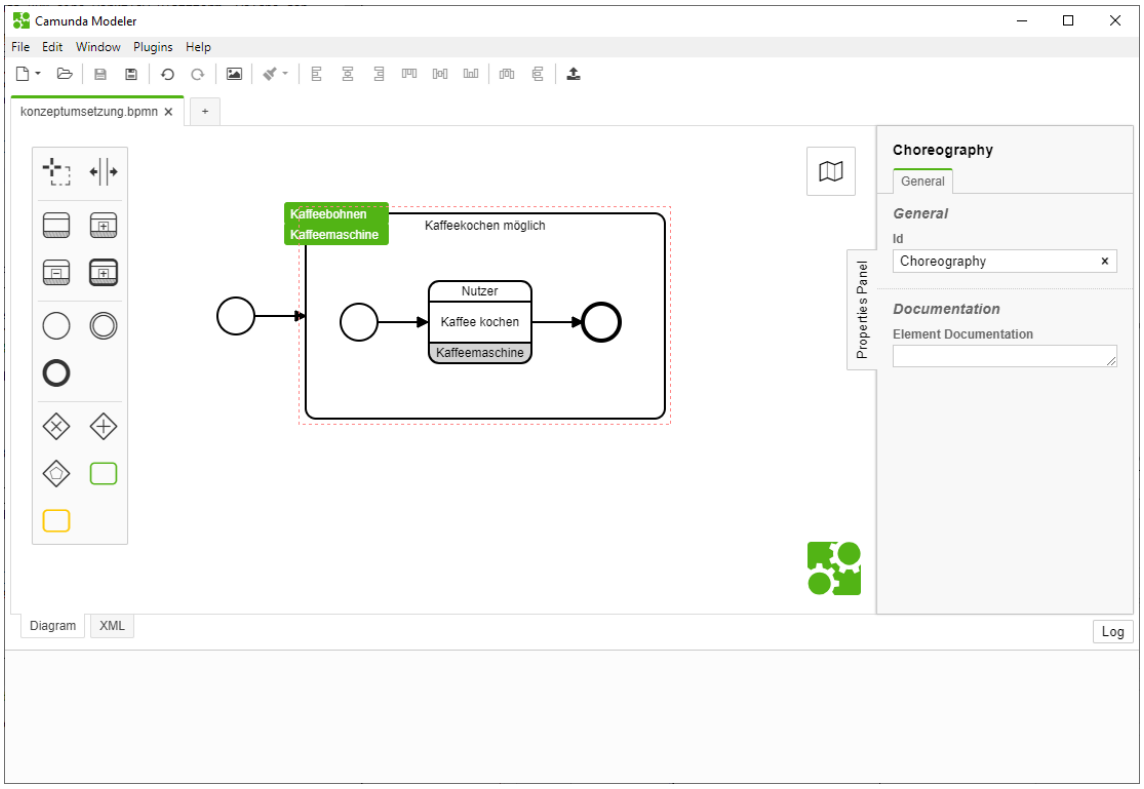


Abbildung 6.2: Camunda Modeler mit der Situational Scope-Erweiterung

Listing 6.4 Erweitern des PropertiesPanelProviders

```
...
import createSituationScopeGroup from './parts/SituationalScopeProperties';
...
function createSituationScopeTab(element,bpmnFactory,moddle) {

  var situationScopeGroup = {
    id: 'situational-scope',
    label: 'Situational Scope',
    entries: []
  };


  createSituationScopeGroup(situationScopeGroup, element,bpmnFactory,moddle);

  return [
    situationScopeTab
  ];
}
  ...

  export default function CustomPropertiesProvider(eventBus, bpmnFactory, canvas,
  elementRegistry, translate, moddle){
    PropertiesActivator.call(this, eventBus);

    this.getTabs = function(element) {
  ...

      var situationScopeTab = {
        id: 'situationproperties',
        label: 'Situation Properties',
        groups: createSituationScopeTab(element,bpmnFactory,moddle)
      };
  ...

      return [
  ...
        situationScopeTab,
  ...
      ];
    };
  }
```

Listing 6.5 Erstellung einer Properties-Gruppe

```
export default function(group, element, bpmnFactory,moddle) {
  var entryselectOptions = [
    { value: 'Abort', name: 'Abort' },
    { value: 'Adapt', name: 'Adapt' },
    { value: 'Return', name: 'Return' },
    { value: 'Retry', name: 'Retry' },
    { value: 'Continue', name: 'Continue' }
  ];
  ...
  group.entries.push(entryFactory.selectBox({
    id : 'entryConditionsvalue',
    description : 'Entry Condition',
    label : 'Choose Entry Condition',
    selectOptions: entryselectOptions,
    modelProperty : 'entryCondition'
  }));
  ...
}
```

Das zweite Plugin implementiert die Transformation. Es lässt sich in jedem Camunda Modeler einbinden. Es transformiert das Situational Scope-Modell in ein normales BPMN-Kollaborationsmodell und zeigt dieses direkt an. Das Situational Scope-Modell wird als JSON-Objekt eingelesen. Mittels Tiefensuche wird das Objekt durchiteriert. Anhand der definierter Situational Scope-Strukturen werden entsprechend dem Konzept aus dem gleichnamigen Kapitel die entsprechenden Kollaborationsdiagramm-Strukturen erstellt und verbunden. Dafür wird das *bpmn-js-cli*-Module verwendet [Gmb20a]. Dieses Interface erstellt sowohl das Modell als auch die visuelle Darstellung des Modells. Allerdings kann man mit diesem Interface nicht alle Elemente direkt erstellen, weshalb man auch andere Modules verwenden muss, um dies zu ermöglichen. Der Codeausschnitt in Listing 6.6 zeigt ein solches Beispiel. Dort wird ein Subprozess erstellt. Mittels bpmn-js-cli wird dabei das Subprozess-Objekt erstellt, abhängig von seinem Elternobjekt. Das Elternobjekt könnte beispielsweise ein Pool sein. Über bpmn-js-cli wird jedoch nur ein kollabierter Subprozess erstellt. Ein kollabierter Subprozess ist ein Subprozess, welcher den inneren Prozess nicht anzeigt. Der Editor erlaubt auch nicht, den inneren Prozess zu modellieren, während der Subprozess kollabiert ist. Deshalb muss das “bpmnReplace“-Modul geladen werden, um das entsprechende “isExpanded“-Attribut zu setzen, welches den Subprozess expandiert und die Modellierung des inneren Prozesses ermöglicht. Auch das Setzen von bestimmten Typen von Events wird auf diese Weise umgesetzt.

Listing 6.6 Erstellung eines Subprozesses

```
var subprocess = this.cli.create('bpmn:SubProcess', {
  x: 70,
  y: 70
  }, parent);
this.bpmnReplace.replaceElement(this.cli.element(subprocess), {
  type: "bpmn:SubProcess",
  isExpanded: true

});
```

7 Diskussion

Das Konzept wurde in einem Prototyp umgesetzt. Es wurde das Modell aus Abbildung 4.1 unter Nutzung dieses Prototyps modelliert. Das sich daraus ergebende Modell sieht man in Abbildung 7.1, welches jetzt näher beschrieben wird.

Die einzelnen Situational Scopes werden als Kästen dargestellt. Situationselemente werden als kleine Kästen an der oberen linken Ecke dieser Kästen dargestellt, deren Werte farblich markiert sind (Grün=True, Rot=False). Situational Scopes ohne Situationselemente stellen Default-Situationen dar. Adapt-Pfade werden als blaue, Continue-Pfade als grüne Verbindungen dargestellt. Innerhalb der Situational Scopes sind die dazugehörigen Choreographiediagramme zu sehen.

Die Transformation erstellt ein Modell, welches dem in Abbildung 4.1 dargestellten Modell gleicht. In dem Modell in Abbildung 7.1 ist die Kausalität der Situationen leicht ersichtlich. Ebenso wird dem Betrachter direkt signalisiert, was Ausnahmesituationen sind und welche Situational Scopes neue Situationen darstellen. Die Bedingungen für die Situationen, also die Situationselemente und die dazugehörigen Werte können leicht abgelesen werden. Das Hinzufügen neuer Situationen erfolgt durch das Hinzufügen neuer Situational Scopes. Der Rest des Modells muss dafür nicht abgeändert werden. Ähnlich verhält es sich mit dem Austauschen von Situational Scopes: Ein neuer Situational Scope kann erstellt und anstatt eines anderen Situational Scopes mit den vorhergehenden und nachfolgenden Scopes verbunden werden. Auch in dem Fall, dass sich nicht die Situation ändert, sondern der technische Prozess, der in der Situation ausgeführt werden soll, sind Änderungen leicht durchführbar: Es muss lediglich das einzelne Choreographiediagramm des dazugehörigen Situational Scopes abgeändert werden.

Schwierigkeiten können bei der Wahl der Interaktionspartner zwischen Situational Scopes entstehen. Wenn nicht mindestens ein Teilnehmer des letzten Choreography-Tasks eines Situational Scopes der initiierende Teilnehmer des ersten Choreography Tasks des nächsten Situational Scopes ist, kann es passieren, dass ein Modell entsteht, welches in einem Deadlock endet. Dies ist ein Nachteil der globalen Sicht auf ein dezentrales System: Der Ablauf wird global definiert, aber Informationen werden nicht automatisch global geteilt. Ein Informationsaustausch findet explizit statt und wenn ein initiierender Teilnehmer nicht vorher an einer Interaktion beteiligt gewesen ist, kann es sein, dass diesem Teilnehmer Informationen fehlen, wann sein Send Task getriggert werden soll. Dies ist jedoch auch ein grundsätzliches Problem bei Choreographiediagrammen, was man mittels Modellierungseinschränkungen lösen könnte. Ein weiteres Problem ist, dass die Kompensation bei Running Compensate Condition nur für den ganzen Ausführungsprozess definiert wird. Es wird außer Acht gelassen, welche Tasks bereits ausgeführt worden sind und welche nicht, weshalb man gezwungen ist, einen relativ generischen Kompensationsprozess zu beschreiben. Bei parallel ablaufenden Prozessen, die mit dem “Interrupting“-Wert definiert worden sind, fehlt eine Kompensationslösung ganz. Eine bessere Lösung, vielleicht unter Verwendung von Compensation Spheres [LR00], könnte in zukünftigen Arbeiten angegangen werden.

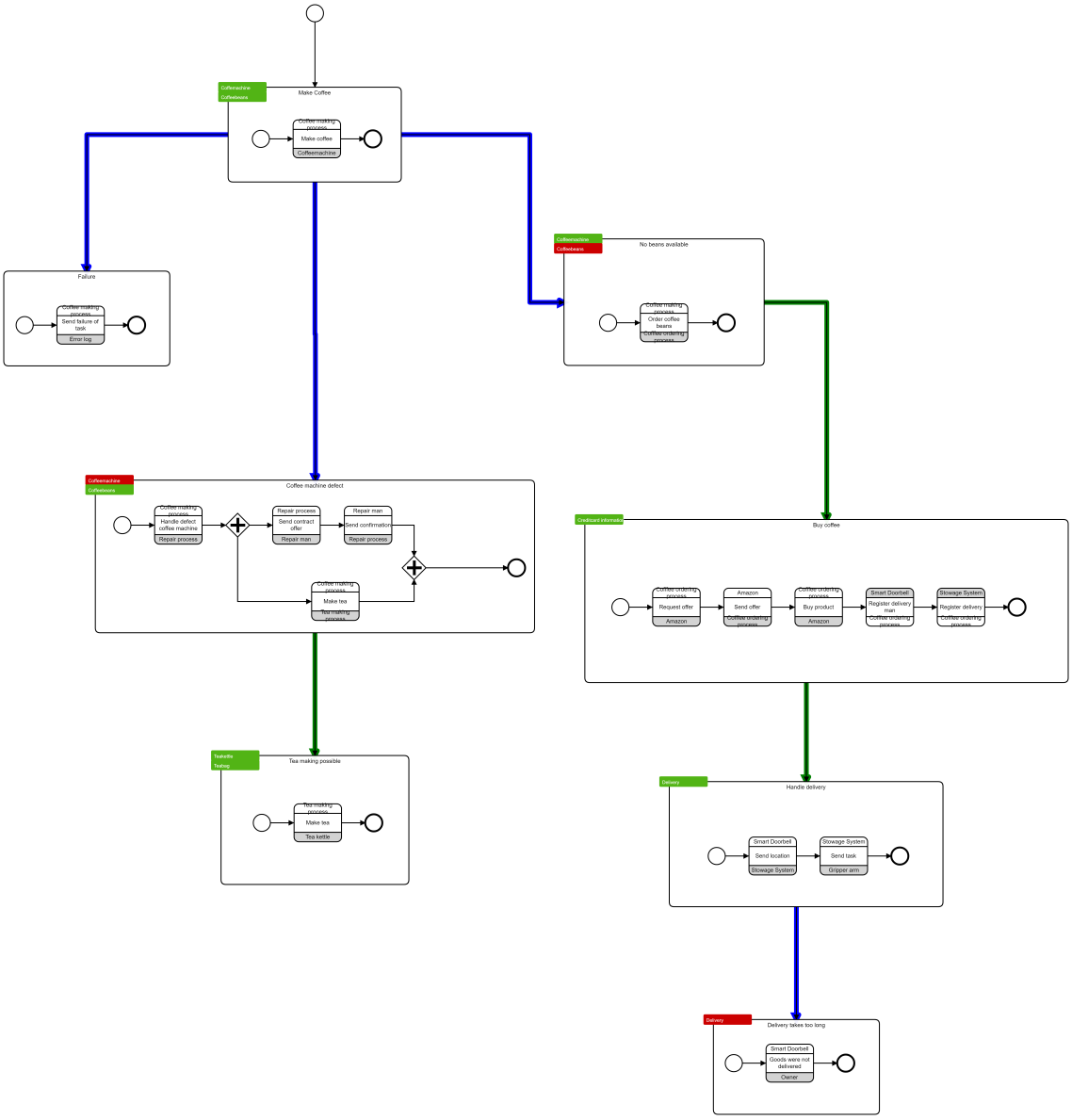


Abbildung 7.1: Umsetzung des in Abbildung 4.1 dargestellten “Kaffee-Kochen“-Prozesses als Situational Scope-Diagramm

8 Zusammenfassung und Ausblick

Die technologische Entwicklung bringt komplexe vernetzte Systeme. Neue Sensoren und Aktuatoren ermöglichen in Kombination mit den vernetzten Systemen neue Anwendungsgebiete, wie etwa situationsadaptive Systeme. Diese können auf Ereignisse und Änderungen von Situationen reagieren und passende Aktionen ausführen. Diese komplexen Systeme zu entwerfen ist sehr schwierig. Viele Geräte müssen integriert werden, viele Situationen müssen im Programmablauf implementiert werden. Hilfreich dabei sind workflow-basierte Modellierungssprachen wie BPMN. Choreographiediagramme aus BPMN können genutzt werden, um die Interaktion zwischen den verschiedenen Geräten zu definieren. Jedoch bleibt es kompliziert, Situationen in solchen Diagrammen abzubilden, da diese Diagramme Situationen nicht explizit abbilden können. Darum ist ein Modellierungsansatz, welches Situationen als Modellierungselement verwendet, notwendig.

Der in dieser Thesis ausgearbeitete Modellierungsansatz orientiert sich an existierenden Ansätzen. Ähnlich wie in SitOpt [WSBL15] wird in diesem Ansatz ein BPMN-Workflow-Modell erstellt, in welchem Adaptionslogik und Workflow-Kontrollfluss unabhängig voneinander modelliert werden können. Dieses Modell wird dann in ein standardkonformes BPMN-Kollaborationsdiagramm umgewandelt. Dabei werden die Choreographiediagramme in Kollaborationsdiagramme und die Situationslogik in Verbindungsstrukturen zwischen den Teilnehmern transformiert. Weitere Einflüsse stammen aus SitME [BHK+15]. Das Konzept von Situational Scopes wurde für das Modellierungskonzept dieser Thesis übernommen und weiter ausgearbeitet. Unter anderem wurden weitere Behandlungsmöglichkeiten für Exceptions hinzugefügt, welche sich mit den Exception Handling-Konzepten von Reichert et al. [RW12] überdecken. Für die Transformation der Behandlungsmöglichkeiten wurde auf die Service Interaction Patterns von Weske et al. [Wes19] zurückgegriffen. Um die Choreographiediagramme des Modellierungskonzepts in Kollaborationen umzuwandeln, wurde ein Transformationsalgorithmus entworfen. Dabei wurden die Überlegungen von Bischoff et al. [BFR19] und Barros et al. [BHF10] beachtet.

Das ganze Konzept wurde in einem Prototyp umgesetzt. Dafür wurde der Open Source BPMN-Editor Camunda Modeler verwendet [Gmb20c]. Erweitert wurde er mit dem chor-js-Framework, welches die Modellierung von Choreographiediagrammen implementiert [LWW19].

Ausblick

Das in dieser Thesis ausgearbeitete Konzept stellt lediglich einen ersten Schritt in der Entwicklung von Choreographie-Modellierungsansätzen für situationsadaptive Systeme dar. Eine Frage, die in dieser Thesis nicht beleuchtet wurde, ist, auf welche Weise die Teilnehmer die für die Evaluation benötigten Informationen bekommen. Eine mögliche Lösung könnte die Einbindung von speziellen Evaluationsprozessen sein, welche diesen Vorgang definieren. Alternativ könnte man dies eventuell mithilfe eines Service Bus realisieren. Für die tatsächliche Verwendung des Modellierungsansatzes ist dies jedoch eines der wichtigsten Probleme, welches gelöst werden muss.

Ein anderes Problem wurde bereits im Diskussionskapitel angesprochen. Die Kompensation von bereits laufenden Prozessen ist viel zu grob definiert und nimmt keinen Bezug auf bereits ausgeführte Tasks. Die Entwicklung eines besseren Konzepts wäre eine mögliche zukünftige Arbeit.

Eine weiterer interessanter Ansatz wäre die Einbindung eines Workflow-Repositories. Mit solch einem könnte man die Modellierung der Ausführungsprozesse vereinfachen oder vollständig obsolet werden lassen und sich hauptsächlich noch auf die Situationsmodellierung beschränken, da man dann vorgefertigte Workflow-Fragmente aus dem Repository in die Situational Scopes einbinden könnte. Eventuell könnten diese Workflow-Fragmente sogar schon technische Prozesse oder einzelne Prozessschritte bzw. Tasks enthalten, womit der Anreicherungsschritt wegfallen und man direkt ein ausführbares Modell erhalten würde.

Literaturverzeichnis

[AH02] W. M. P. van der Aalst, K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. Cooperative information systems. MIT Press, 2002. ISBN: 0-262-01189-1 (zitiert auf S. 23, 24).

[AHEA05] M. Adams, A. H. M. ter Hofstede, D. Edmond, W. M. P. van der Aalst. „Facilitating Flexibility and Dynamic Exception Handling in Workflows through Worklets“. In: *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05), Porto, Portugal, 13-17 June, 2005, CAiSE Forum, Short Paper Proceedings*. 2005. URL: http://ceur-ws.org/Vol-161/FORUM%5C_08.pdf (zitiert auf S. 30).

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, C. E. Landwehr. „Basic Concepts and Taxonomy of Dependable and Secure Computing“. In: *IEEE Trans. Dependable Sec. Comput.* 1.1 (2004), S. 11–33. DOI: 10.1109/TDSC.2004.2. URL: https://doi.org/10.1109/TDSC.2004.2 (zitiert auf S. 30).

[BBK+13] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, J. Wettinger. „Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies“. In: *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings*. 2013, S. 130–148. DOI: 10.1007/978-3-642-41030-7_9. URL: https://doi.org/10.1007/978-3-642-41030-7%5C_9 (zitiert auf S. 18).

[BFR19] F. Bischoff, W. Fdhila, S. Rinderle-Ma. „Generation and Transformation of Compliant Process Collaboration Models to BPMN“. In: *Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings*. 2019, S. 462–478. DOI: 10.1007/978-3-030-21290-2_29. URL: https://doi.org/10.1007/978-3-030-21290-2%5C_29 (zitiert auf S. 27–29, 42, 60, 79).

[BHF10] A. Barros, T. Hettel, C. Flender. „Process Choreography Modeling“. In: Mai 2010, S. 257–277. DOI: 10.1007/978-3-642-00416-2_12 (zitiert auf S. 42, 79).

[BHK+15] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, M. Wieland. „A situation-aware workflow modelling extension“. In: *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2015, Brussels, Belgium, December 11-13, 2015*. 2015, 64:1–64:7. DOI: 10.1145/2837185.2837248. URL: https://doi.org/10.1145/2837185.2837248 (zitiert auf S. 39, 40, 50, 51, 79).

[Dey01] A. K. Dey. „Understanding and Using Context“. In: *Personal and Ubiquitous Computing* 5.1 (2001), S. 4–7. DOI: 10.1007/s007790170019. URL: https://doi.org/10.1007/s007790170019 (zitiert auf S. 39).

[Dou04] P. Dourish. „What we talk about when we talk about context“. In: *Personal and Ubiquitous Computing* 8.1 (2004), S. 19–30. DOI: 10.1007/s00779-003-0253-8. URL: https://doi.org/10.1007/s00779-003-0253-8 (zitiert auf S. 39).

[DW11] G. Decker, M. Weske. „Interaction-centric modeling of process choreographies“. In: *Inf. Syst.* 36.2 (2011), S. 292–312. DOI: 10.1016/j.is.2010.06.005. URL: https://doi.org/10.1016/j.is.2010.06.005 (zitiert auf S. 27).

[EL95] J. Eder, W. Liebhart. „The Workflow Activity Model WAMO“. In: *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS-95), May 9-12, 1995, Schloss Wilhelminenburg Hotel, Vienna, Austria*. 1995, S. 87–98 (zitiert auf S. 31).

[Fou20] O. Foundation. *About Node.js*. Version: Februar 2020. URL: https://nodejs.org/en/about/ (zitiert auf S. 68).

[FR14] J. Freund, B. Rücker. *Praxishandbuch BPMN 2.0*. Carl Hanser Verlag GmbH Co KG, 2014 (zitiert auf S. 25).

[Gmb20a] C. S. GmbH. *bpmn-js-cli*. Version: Februar 2020. URL: https://github.com/bpmn-io/bpmn-js-cli (zitiert auf S. 75).

[Gmb20b] C. S. GmbH. *BPMN-Moddle*. Version: Februar 2020. URL: https://github.com/bpmn-io/bpmn-moddle (zitiert auf S. 69).

[Gmb20c] C. S. GmbH. *Camunda Modeler*. Version: Februar 2020. URL: https://camunda.com/products/modeler/ (zitiert auf S. 67, 79).

[Gmb20d] C. S. GmbH. *Camunda Modeler*. Version: Februar 2020. URL: https://github.com/camunda/camunda-modeler (zitiert auf S. 67).

[Gmb20e] C. S. GmbH. *Camunda Modeler Developer Documentation*. Version: Februar 2020. URL: https://github.com/camunda/camunda-modeler/tree/master/docs (zitiert auf S. 67).

[Gmb20f] C. S. GmbH. *Camunda Modeler Documentation*. Version: Februar 2020. URL: https://docs.camunda.org/manual/7.4/modeler/camunda-modeler/ (zitiert auf S. 67).

[Gre01] S. Greenberg. „Context as a Dynamic Construct“. In: *Human-Computer Interaction* 16.2-4 (2001), S. 257–268. DOI: 10.1207/S15327051HCI16234_09. URL: https://doi.org/10.1207/S15327051HCI16234%5C_09 (zitiert auf S. 19, 39, 51).

[HBM08] R. Hamadi, B. Benatallah, B. Medjahed. „Self-adapting recovery nets for policy-driven exception handling in business processes“. In: *Distributed and Parallel Databases* 23.1 (2008), S. 1–44. DOI: 10.1007/s10619-007-7020-1. URL: https://doi.org/10.1007/s10619-007-7020-1 (zitiert auf S. 31).

[HC09] M. Hammer, J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. Zondervan, 2009 (zitiert auf S. 23).

[HH95] D. Hollingsworth, U. Hampshire. „Workflow management coalition: The workflow reference model“. In: *Document Number TC00-1003* 19 (1995), S. 16 (zitiert auf S. 23).

[HHBS18] M. T. Hammi, B. Hammi, P. Bellot, A. Serhrouchni. „Bubbles of Trust: A decentralized Blockchain-based authentication system for IoT“. In: *Computers & Security* 78 (2018), S. 126–142. DOI: 10.1016/j.cose.2018.06.004. URL: https://doi.org/10.1016/j.cose.2018.06.004 (zitiert auf S. 19).

[IAC16] U. A. Ibarra, J. C. Augusto, T. Clark. „Engineering context-aware systems and applications: A survey“. In: *Journal of Systems and Software* 117 (2016), S. 55–83. DOI: 10.1016/j.jss.2016.02.010. URL: https://doi.org/10.1016/j.jss.2016.02.010 (zitiert auf S. 39).

[Inc20a] G. Inc. *Electron-Dokumentation*. Version: Februar 2020. URL: https://www.electronjs.org/docs/tutorial/first-app (zitiert auf S. 68).

[Inc20b] G. Inc. *Node.js Modules*. Version: February 2020. URL: https://nodejs.org/api/modules.html (zitiert auf S. 69).

[Kha10] R. Khadka. „Model-Driven Development of Service Compositions: Transformation from Service Choreography to Service Orchestrations“. Magisterarb. University of Twente, 2010 (zitiert auf S. 19).

[LR00] F. Leymann, D. Roller. *Production Workflow - Concepts and Techniques*. Prentice Hall, 2000. ISBN: 978-0-13-021753-0 (zitiert auf S. 23, 35, 59, 77).

[LWW19] J. Ladleif, A. von Weltzien, M. Weske. „chor-js: A Modeling Framework for BPMN 2.0 Choreography Diagrams“. In: *Proceedings of the ER Forum and Poster & Demos Session 2019 on Publishing Papers with CEUR-WS co-located with 38th International Conference on Conceptual Modeling (ER 2019), Salvador, Brazil, November 4, 2019*. 2019, S. 113–117. URL: http://ceur-ws.org/Vol-2469/ERDemo02.pdf (zitiert auf S. 69, 79).

[MD17] F. Martins, D. Domingos. „Modelling IoT behaviour within BPMN Business Processes“. In: *Procedia Computer Science* 121 (Jan. 2017), S. 1014–1022. DOI: 10.1016/j.procs.2017.11.131 (zitiert auf S. 17).

[Mey13] D. Meyer. *camunda forks Activiti and launches camunda BPM*. 2013. URL: https://blog.camunda.com/post/2013/03/camunda-forks-activiti-and-launches/ (zitiert auf S. 67).

[MH08] J. Mendling, M. Hafner. „From WS-CDL Choreography to BPEL Process Orchestration“. In: *Journal of Enterprise Information Management* (2008) (zitiert auf S. 19).

[MSPZ18] J. Mocnej, W. K. Seah, A. Pekar, I. Zolotova. „Decentralised IoT Architecture for Efficient Resources Utilisation“. In: *IFAC-PapersOnLine* 51.6 (2018), S. 168–173 (zitiert auf S. 19).

[OMG13] OMG. *Business Process Model and Notation (BPMN), Version 2.0.2*. Object Management Group, Dez. 2013. URL: http://www.omg.org/spec/BPMN/2.0.2 (zitiert auf S. 24–26, 29).

[PCS18] J. Ploennigs, J. Cohn, A. J. Stanford-Clark. „The Future of IoT“. In: *IEEE Internet Things Mag.* 1.1 (2018), S. 28–33. DOI: 10.1109/IOTM.2018.1700021. URL: https://doi.org/10.1109/IOTM.2018.1700021 (zitiert auf S. 19).

[RGHH13] S. Rebai, N. Guermouche, H. Hadj Kacem, A. Hadj Kacem. „Towards Error-Handling-aware Choreography to Orchestration Transformation Approach“. In: *Int. J. of Collaborative Enterprise* 3 (Jan. 2013), S. 151–166. DOI: 10.1504/IJCENT.2013.053293 (zitiert auf S. 27).

[RW12] M. Reichert, B. Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012. ISBN: 978-3-642-30408-8. DOI: 10.1007/978-3-642-30409-5. URL: https://doi.org/10.1007/978-3-642-30409-5 (zitiert auf S. 29, 31–37, 52, 79).

[SNPB14] C. Sarkar, S. N. A. U. Nambi, R. V. Prasad, A. R. Biswas. „A Scalable Distributed Architecture Towards Unifying IoT Applications“. In: *IEEE World Forum on Internet of Things, WF-IoT 2014, Seoul, South Korea, March 6-8, 2014*. 2014, S. 508–513. DOI: 10.1109/WF-IoT.2014.6803220. URL: https://doi.org/10.1109/WF-IoT.2014.6803220 (zitiert auf S. 19).

[SO00] S. Sadiq, M. Orlowska. „On Capturing Exceptions in Workflow Process Models“. In: (Jan. 2000). DOI: 10.1007/978-1-4471-0761-3_1 (zitiert auf S. 31).

[Wes19] M. Weske. *Business Process Management - Concepts, Languages, Architectures, Third Edition*. Springer, 2019. ISBN: 978-3-662-59431-5. DOI: 10.1007/978-3-662-59432-2. URL: https://doi.org/10.1007/978-3-662-59432-2 (zitiert auf S. 23–25, 27, 43–45, 79).

[WSBL15] M. Wieland, H. Schwarz, U. Breitenbücher, F. Leymann. „Towards Situation-Aware Adaptive Workflows: SitOPT - A General Purpose Situation-Aware Workflow Management System“. In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2015, St. Louis, MO, USA, March 23-27, 2015*. 2015, S. 32–37. DOI: 10.1109/PERCOMW.2015.7133989. URL: https://doi.org/10.1109/PERCOMW.2015.7133989 (zitiert auf S. 41, 79).

[YAH+17] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. Abdallaahmed, A. Gani, M. I. Razzak, M. Guizani. „Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges“. In: *IEEE Wireless Commun.* 24.3 (2017), S. 10–16. DOI: 10.1109/MWC.2017.1600421. URL: https://doi.org/10.1109/MWC.2017.1600421 (zitiert auf S. 17).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift